Day 3 - API Integration Report - Car Rental

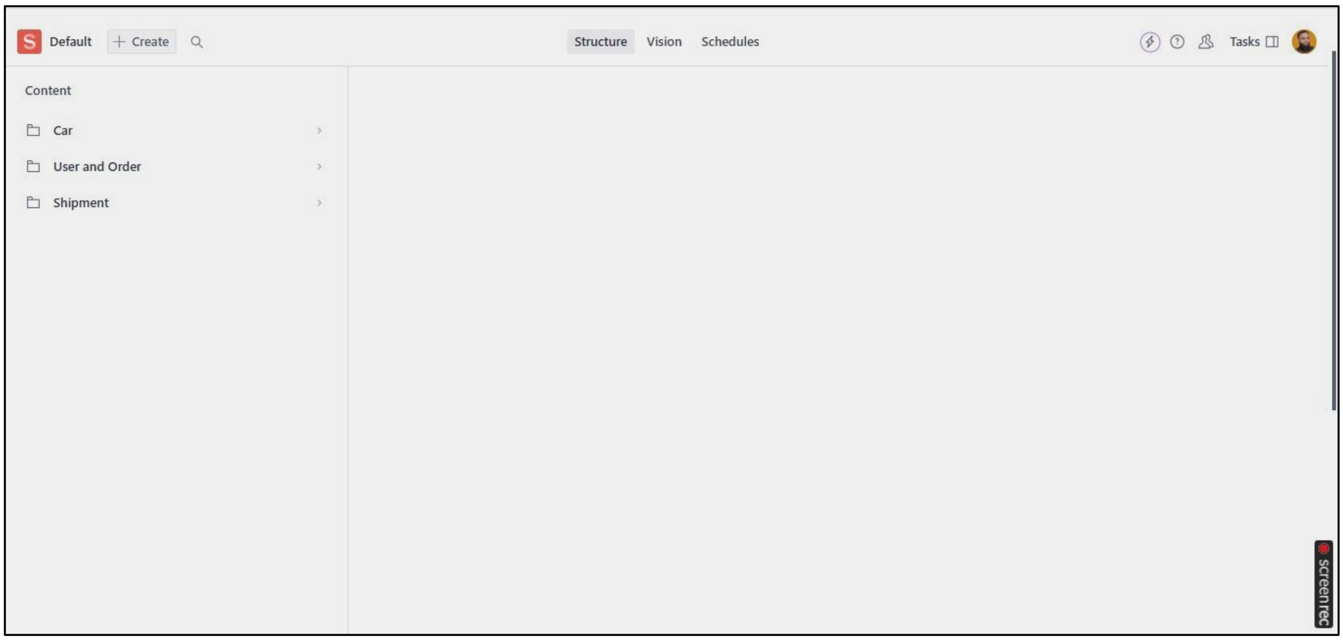
Project Overview

This report documents the API integration process, schema adjustments, and data migration steps performed to populate Sanity CMS with imported data and display it on the front-end. Screenshots and code snippets are included to provide clarity and demonstrate success.

Steps Completed

Step 1: Installing Sanity Studio

1. Installed Sanity Studio into the Next.js project using the command npx sanity init.
2. Verified the installation by accessing the Sanity dashboard.
3. **Outcome**: Successfully set up Sanity Studio in the project structure.

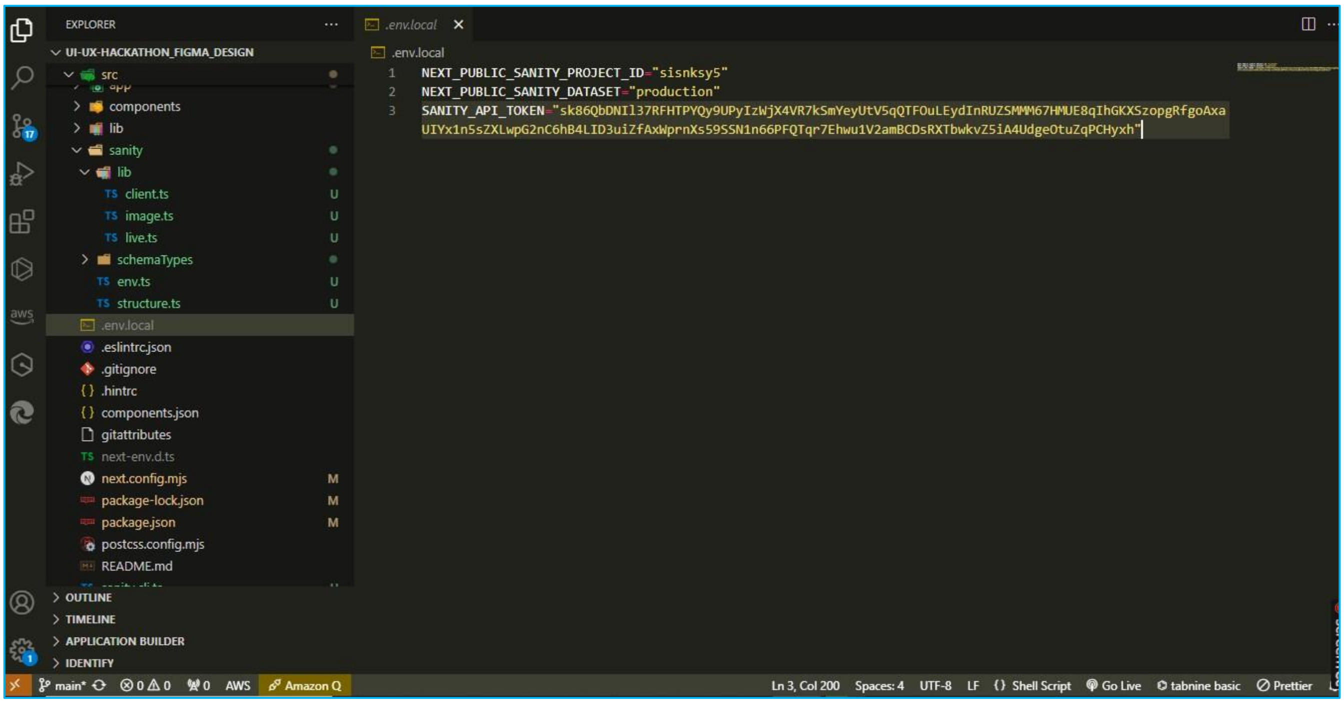


Step 2: Setting Environment Variables

1. Created a .env.local file at the root of the project.
2. Added the following environment variables:

    a. NEXT_PUBLIC_SANITY_PROJECT_ID: Your Sanity project ID.
    b. NEXT_PUBLIC_SANITY_DATASET: Dataset name (e.g., production).
    c. SANITY_API_TOKEN: Token for API authentication.

3. Used the dotenv library to securely load these variables.
4. **Outcome**: Environment variables were successfully configured.



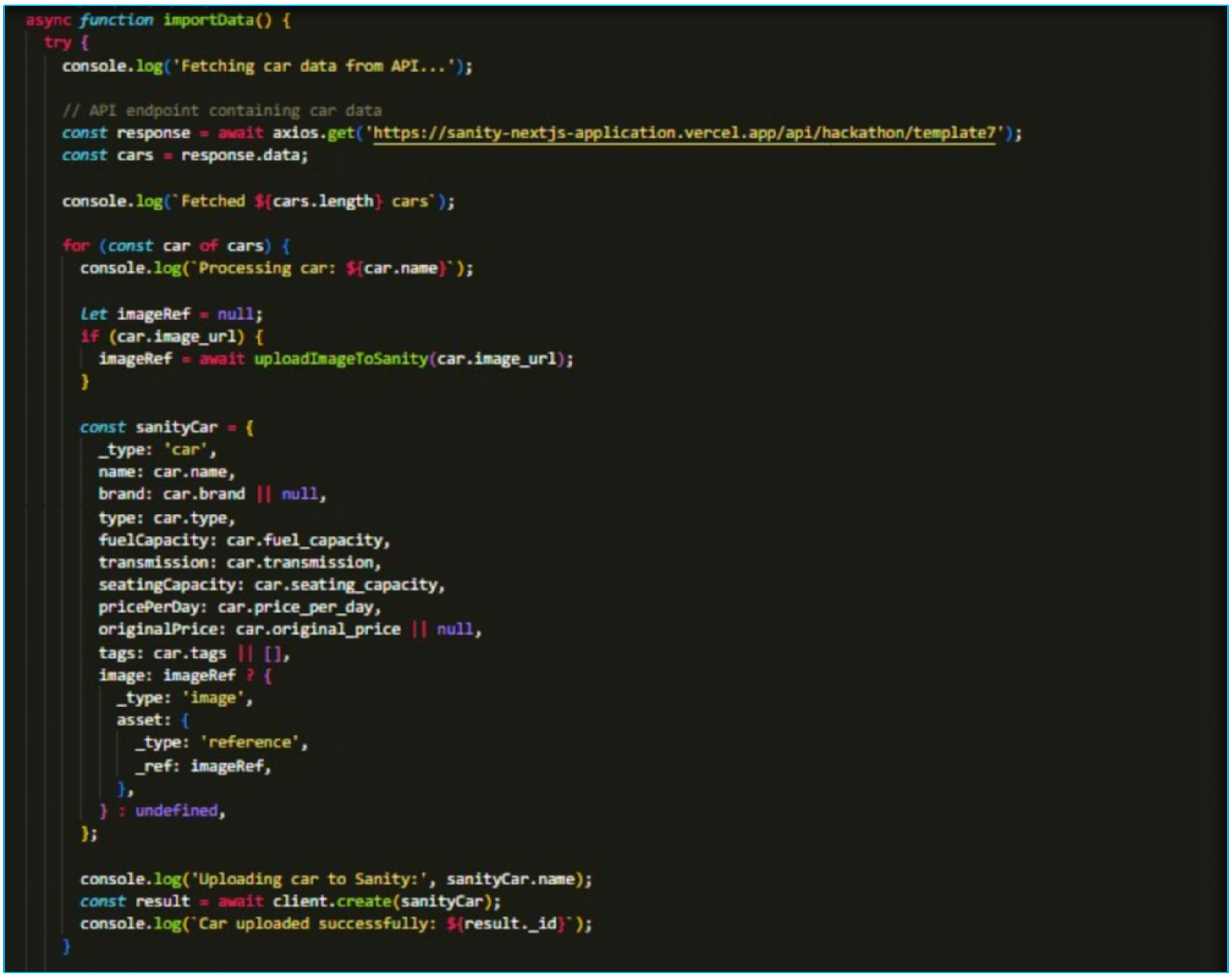
```
NEXT_PUBLIC_SANITY_PROJECT_ID="sisnksy5"
NEXT_PUBLIC_SANITY_DATASET="production"
SANITY_API_TOKEN="sk86QbDNIl37RFHTPYQy9UPyIzWjX4VR7kSmYeyUtV5qQTFOuLEydInRUZSMMM67HMUE8qIhGKXSzopgRfgoAxaUIYx1n5sZXLwpG2nC6hB4LID3uiZfAxWprnXs59SSN1n66PFQTqr7Ehwu1V2amBCDsRXTbwkvZ5iA4UdgeOtuZqPCHyxh"
```

Note: Ensure these variables are not exposed in public repositories for security purposes.

Step 3: Fetching API Data into Sanity Studio

1. **Script Setup:** Created a script importTemplate7Data.ts to handle API data import.

    1. Utilized axios to fetch data from the provided external API.
    2. Utilized @sanity/client to create and upload content to Sanity CMS.



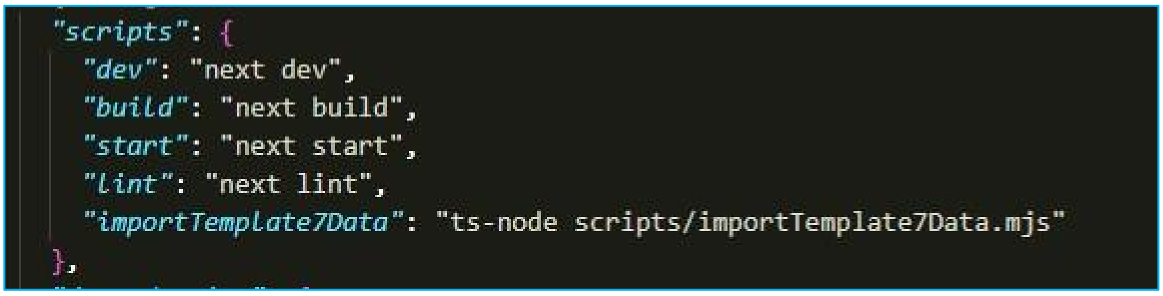
```
async function importData() {
  try {
    console.log('Fetching car data from API...');

    // API endpoint containing car data
    const response = await axios.get('https://sanity-nextjs-application.vercel.app/api/hackathon/template7');
    const cars = response.data;

    console.log(`Fetched ${cars.length} cars`);

    for (const car of cars) {
      console.log(`Processing car: ${car.name}`);

      let imageRef = null;
      if (car.image_url) {
        imageRef = await uploadImageToSanity(car.image_url);
      }

      const sanityCar = {
        _type: 'car',
        name: car.name,
        brand: car.brand || null,
        type: car.type,
        fuelCapacity: car.fuel_capacity,
        transmission: car.transmission,
        seatingCapacity: car.seating_capacity,
        pricePerDay: car.price_per_day,
        originalPrice: car.original_price || null,
        tags: car.tags || [],
        image: imageRef ? {
          _type: 'image',
          asset: {
            _type: 'reference',
            _ref: imageRef,
          },
        } : undefined,
      };

      console.log('Uploading car to Sanity:', sanityCar.name);
      const result = await client.create(sanityCar);
      console.log(`Car uploaded successfully: ${result._id}`);
    }
```

2. **Logic Implementation:**

    1. Implemented a function to upload images and map API data to Sanity-compatible schema types.
    2. Used the create Or Replace method to upload each record into Sanity Studio.

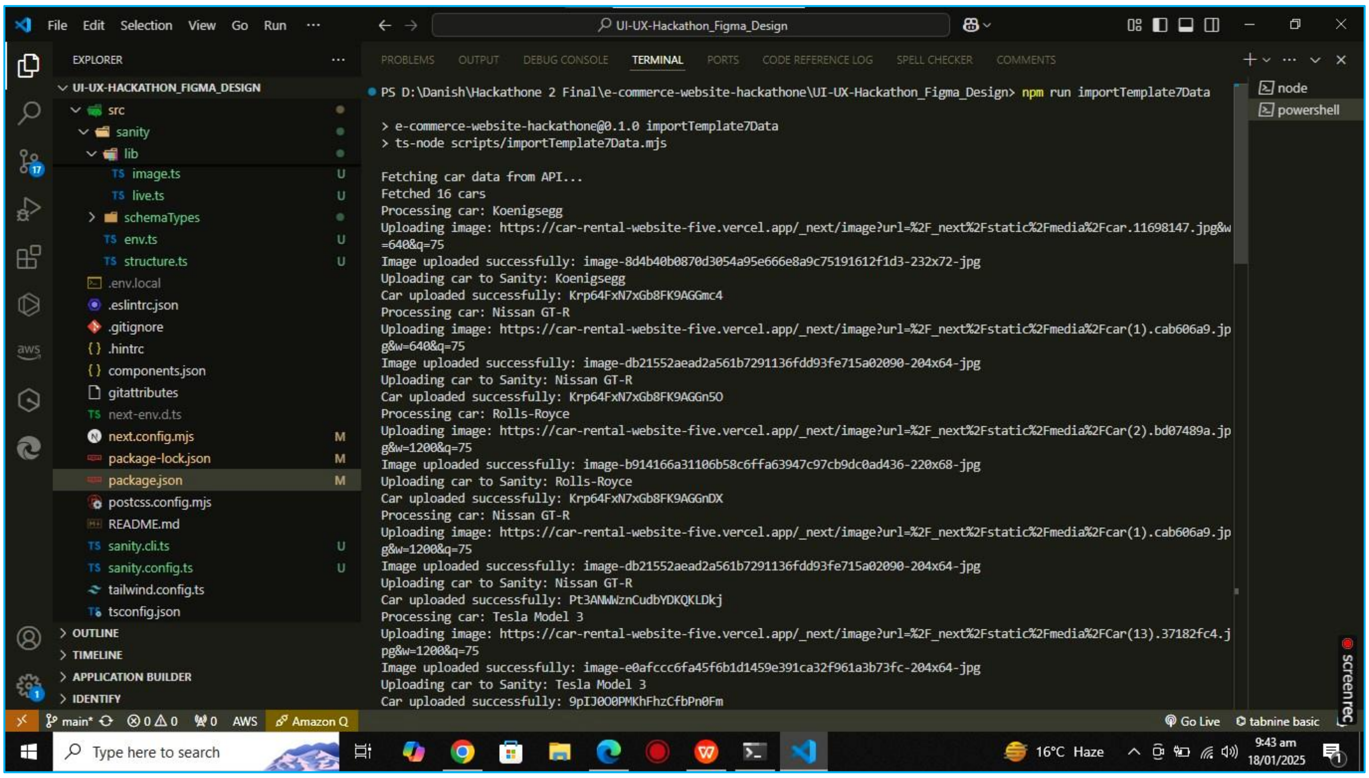
3. **Execution:**

    1. Added a custom script in package.json: "importData": "ts-node importTemplate7Data.ts".



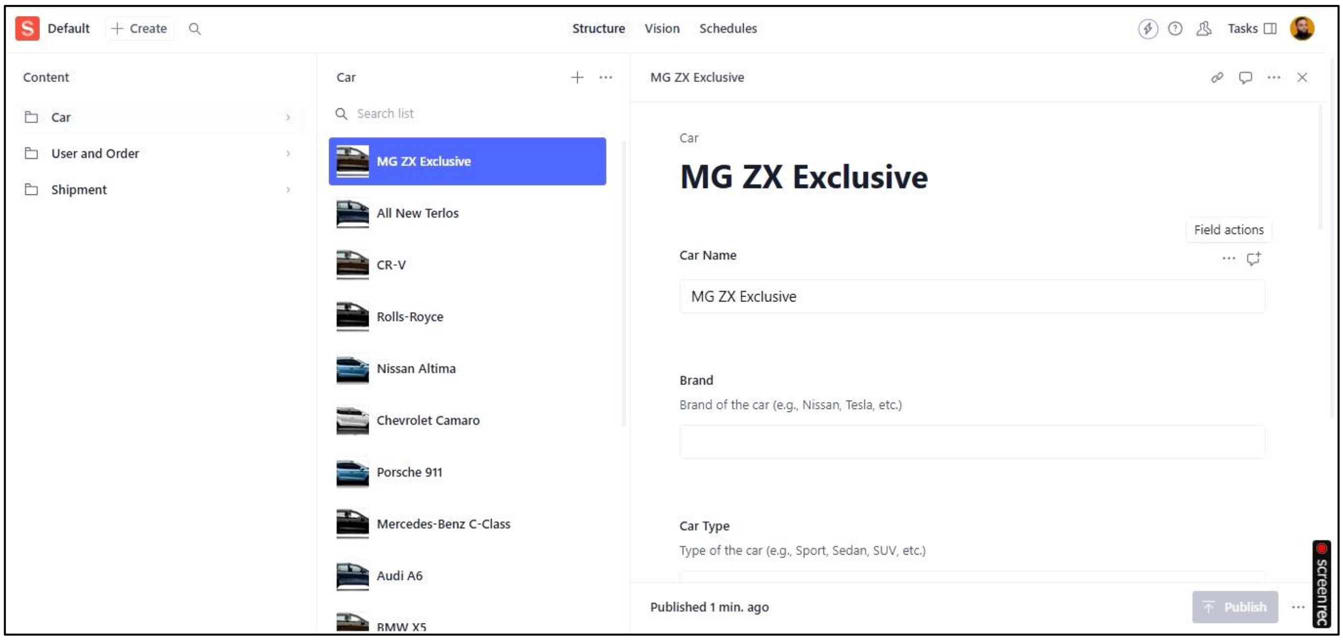
```
"scripts": {
  "dev": "next dev",
  "build": "next build",
  "start": "next start",
  "lint": "next lint",
  "importTemplate7Data": "ts-node scripts/importTemplate7Data.mjs"
},
```

    2. Executed the script with the command npm run importData.



```
PS D:\Danish\Hackathone 2 Final\e-commerce-website-hackathone\UI-UX-Hackathon_Figma_Design> npm run importTemplate7Data

> e-commerce-website-hackathone@0.1.0 importTemplate7Data
> ts-node scripts/importTemplate7Data.mjs

Fetching car data from API...
Fetched 16 cars
Processing car: Koenigsegg
Uploading image: https://car-rental-website-five.vercel.app/_next/image?url=%2F_next%2Fstatic%2Fmedia%2Fcar.11698147.jpg&w
=640&q=75
Image uploaded successfully: image-8d4b40b0870d3054a95e666e8a9c75191612f1d3-232x72-jpg
Uploading car to Sanity: Koenigsegg
Car uploaded successfully: Krp64FxN7xGb8FK9AGGmc4
Processing car: Nissan GT-R
Uploading image: https://car-rental-website-five.vercel.app/_next/image?url=%2F_next%2Fstatic%2Fmedia%2Fcar(1).cab606a9.jp
g&w=640&q=75
Image uploaded successfully: image-db21552aead2a561b7291136fdd93fe715a02090-204x64-jpg
Uploading car to Sanity: Nissan GT-R
Car uploaded successfully: Krp64FxN7xGb8FK9AGGn5O
Processing car: Rolls-Royce
Uploading image: https://car-rental-website-five.vercel.app/_next/image?url=%2F_next%2Fstatic%2Fmedia%2FCar(2).bd07489a.jp
g&w=1200&q=75
Image uploaded successfully: image-b914166a31106b58c6ffa63947c97cb9dc0ad436-220x68-jpg
Uploading car to Sanity: Rolls-Royce
Car uploaded successfully: Krp64FxN7xGb8FK9AGGnDX
Processing car: Nissan GT-R
Uploading image: https://car-rental-website-five.vercel.app/_next/image?url=%2F_next%2Fstatic%2Fmedia%2Fcar(1).cab606a9.jp
g&w=1200&q=75
Image uploaded successfully: image-db21552aead2a561b7291136fdd93fe715a02090-204x64-jpg
Uploading car to Sanity: Nissan GT-R
Car uploaded successfully: Pt3ANWWznCudbYDKQKLDkj
Processing car: Tesla Model 3
Uploading image: https://car-rental-website-five.vercel.app/_next/image?url=%2F_next%2Fstatic%2Fmedia%2FCar(13).37182fc4.j
pg&w=1200&q=75
Image uploaded successfully: image-e0afccc6fa45f6b1d1459e391ca32f961a3b73fc-204x64-jpg
Uploading car to Sanity: Tesla Model 3
Car uploaded successfully: 9pIJ0O0PMKhFhzCfbPn0Fm
```

4. **Outcome:** Successfully imported external data into Sanity Studio.



Car

MG ZX Exclusive

Car Name

MG ZX Exclusive

Brand

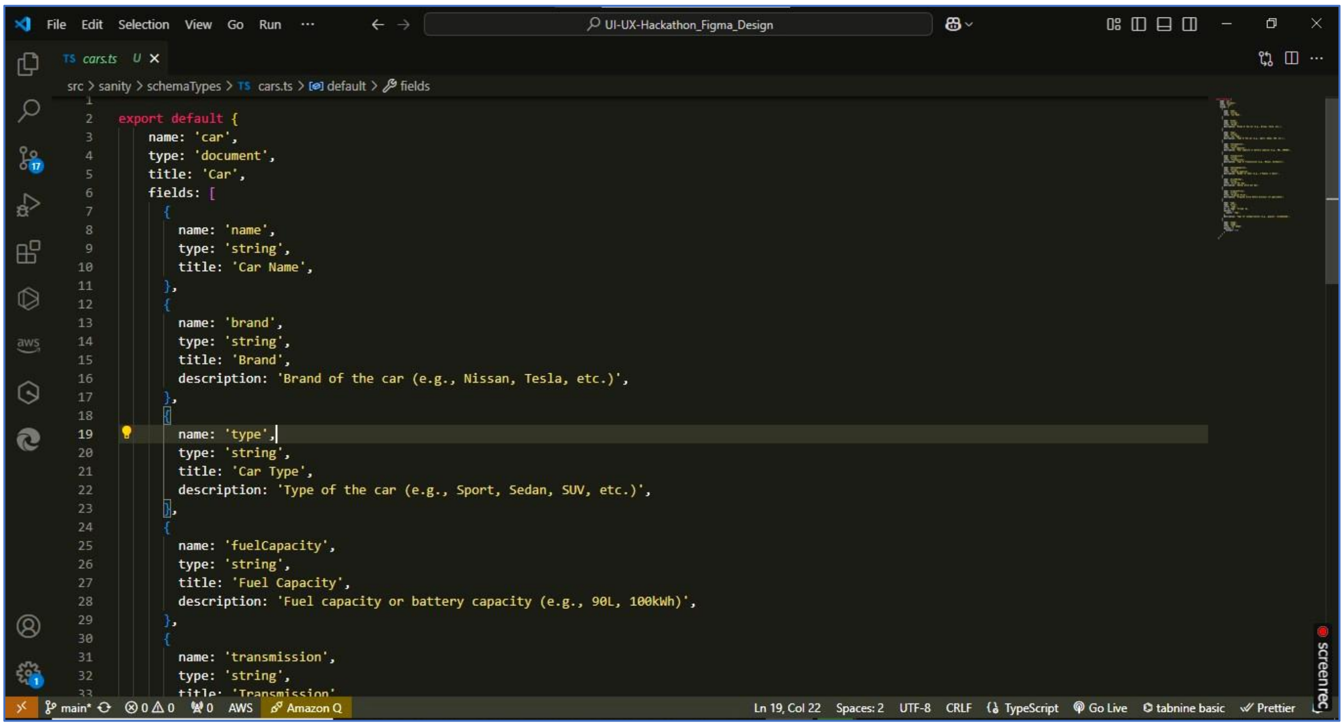
Brand of the car (e.g., Nissan, Tesla, etc.)

Car Type

Type of the car (e.g., Sport, Sedan, SUV, etc.)

Step 4: Adjusting Schemas

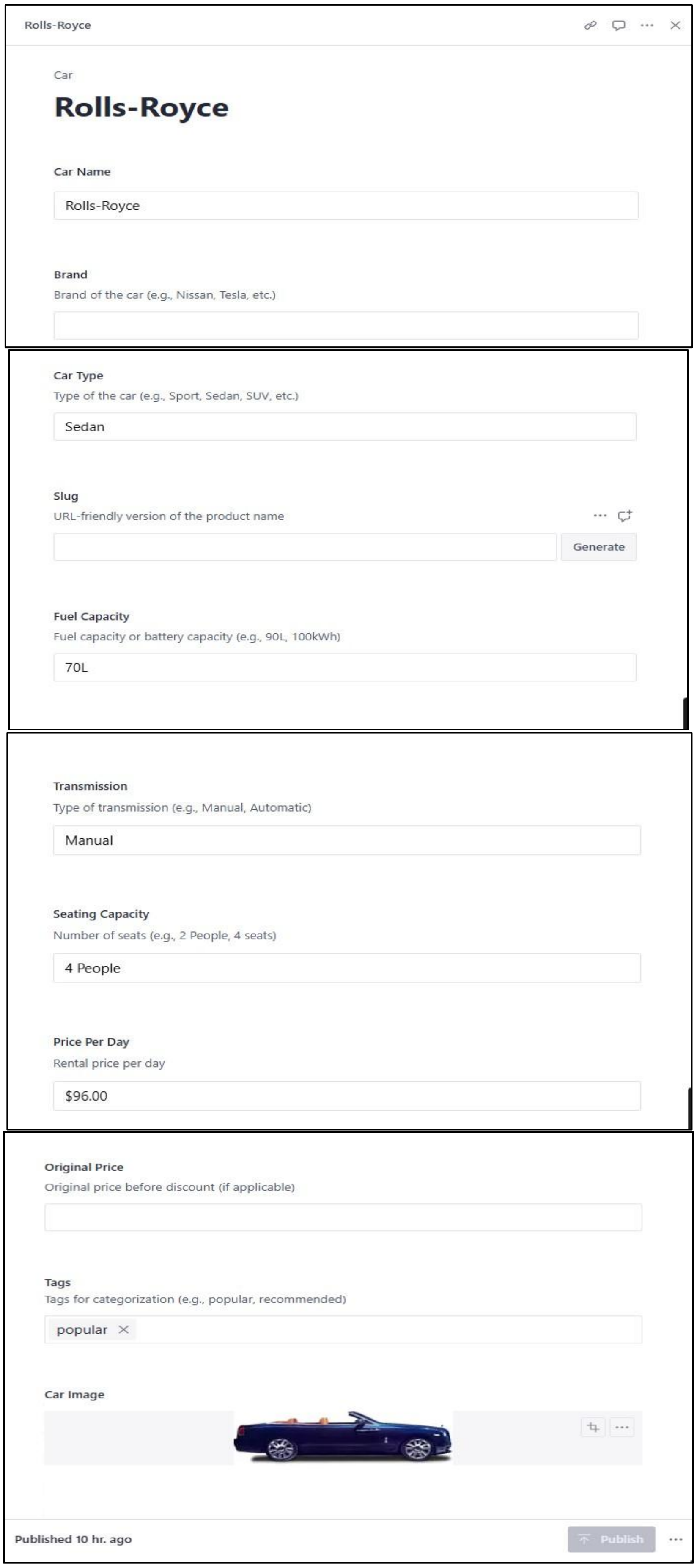
1. Reviewed and updated schemas to match imported data structure (e.g., car type).
2. Added fields such as title, description, price, and image to ensure compatibility.


File Edit Selection View Go Run ···
UI-UX-Hackathon_Figma_Design
cars.ts U
src > sanity > schemaTypes > cars.ts > default > fields
main* ⊗ 0 ⚠ 0 ⚡0 AWS Amazon Q
Ln 19, Col 22 Spaces: 2 UTF-8 CRLF TypeScript Go Live tabnine basic Prettier
screenrec

```

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'
```

3. Validated schema adjustments using Sanity Studio’s preview feature.

4. **Outcome**: Schema successfully aligned with external API data structure.



Rolls-Royce

Car

Rolls-Royce

Car Name

Rolls-Royce

Brand

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

Car Type

Type of the car (e.g., Sport, Sedan, SUV, etc.)

Sedan

Slug

URL-friendly version of the product name

Generate

Fuel Capacity

Fuel capacity or battery capacity (e.g., 90L, 100kWh)

70L

Transmission

Type of transmission (e.g., Manual, Automatic)

Manual

Seating Capacity

Number of seats (e.g., 2 People, 4 seats)

4 People

Price Per Day

Rental price per day

$96.00

Original Price

Original price before discount (if applicable)

Tags

Tags for categorization (e.g., popular, recommended)

popular ×

Car Image

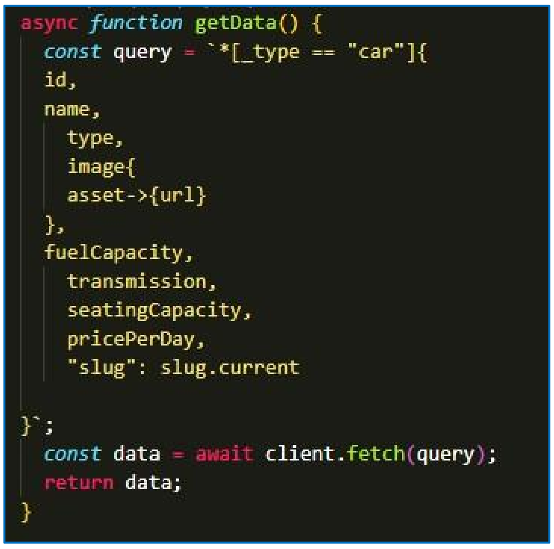
Published 10 hr. ago

Publish

Step 5: Front-end API Integration

1. **Data Fetching:** Used next-sanity to fetch data from Sanity.

    1. Created a query to retrieve car listings and their details.

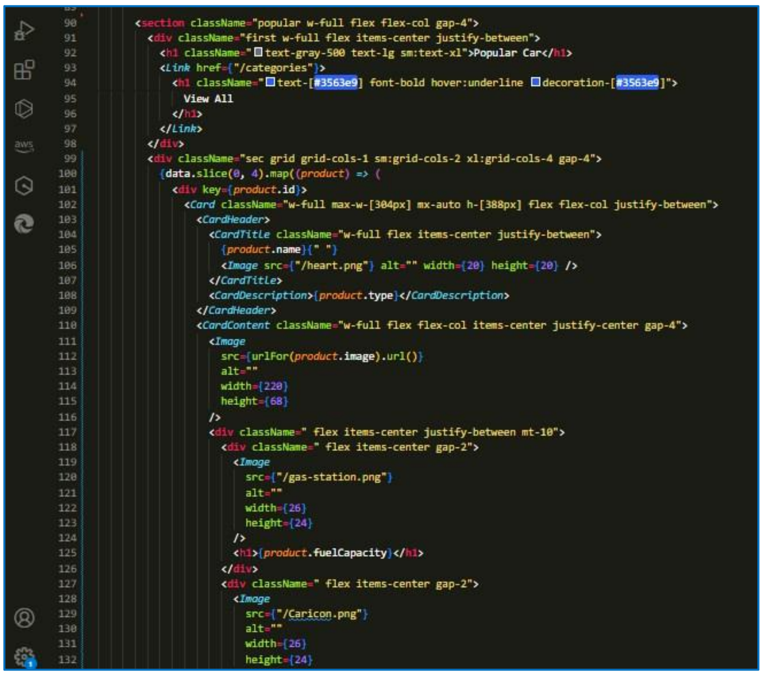


```
async function getData() {
  const query = `*[_type == "car"]{
  id,
  name,
    type,
    image{
    asset->{url}
  },
  fuelCapacity,
    transmission,
    seatingCapacity,
    pricePerDay,
    "slug": slug.current

}`;
  const data = await client.fetch(query);
  return data;
}
```

    2. Verified the API response using the browser console and Postman.

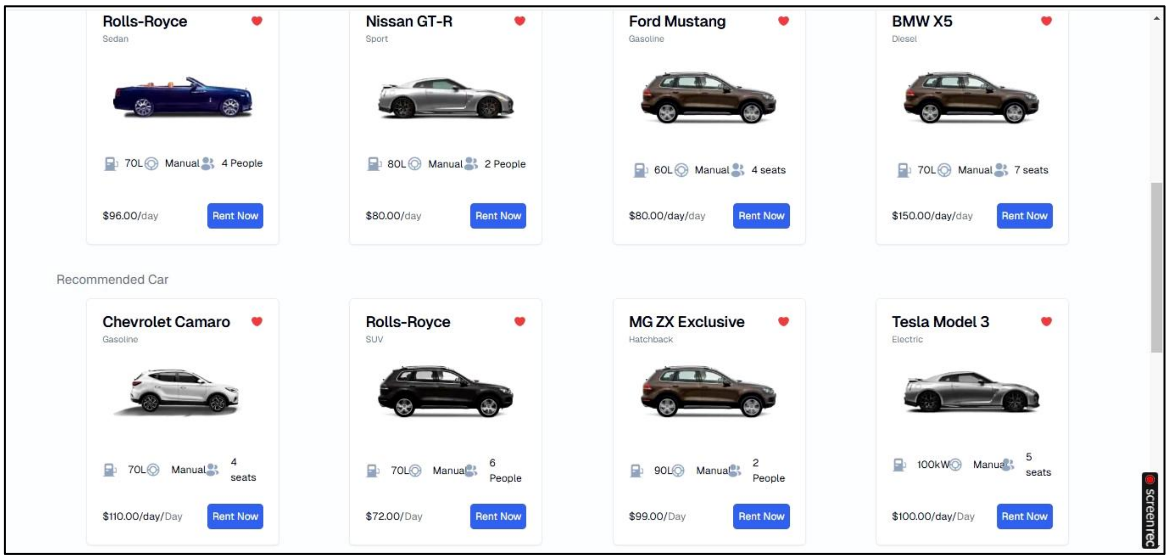
2. **Component Development:**

    1. Designed a grid layout component to display car details (e.g., title, price, image).
    2. Integrated Tailwind CSS for styling.



```
<section className="popular w-full flex flex-col gap-4">
  <div className="first w-full flex items-center justify-between">
    <h1 className="text-gray-500 text-lg sm:text-xl">Popular Car</h1>
    <Link href={"/categories"}>
      <h1 className="text-[#3563e9] font-bold hover:underline decoration-[#3563e9]">
        View All
      </h1>
    </Link>
  </div>
  <div className="sec grid grid-cols-1 sm:grid-cols-2 xl:grid-cols-4 gap-4">
    {data.slice(0, 4).map((product) => (
      <div key={product.id}>
        <Card className="w-full max-w-[304px] mx-auto h-[388px] flex flex-col justify-between">
          <CardHeader>
            <CardTitle className="w-full flex items-center justify-between">
              {product.name}{" "}
              <Image src={"/heart.png"} alt="" width={20} height={20} />
            </CardTitle>
            <CardDescription>{product.type}</CardDescription>
          </CardHeader>
          <CardContent className="w-full flex flex-col items-center justify-center gap-4">
            <Image
              src={urlFor(product.image).url()}
              alt=""
              width={220}
              height={68}
            />
            <div className=" flex items-center justify-between mt-10">
              <div className=" flex items-center gap-2">
                <Image
                  src={"/gas-station.png"}
                  alt=""
                  width={26}
                  height={24}
                />
                <h1>{product.fuelCapacity}</h1>
              </div>
              <div className=" flex items-center gap-2">
                <Image
                  src={"/Caricon.png"}
                  alt=""
                  width={26}
                  height={24}
```

3. **Testing:** Verified data rendering on the front-end and ensured no layout or API errors.
4. **Outcome**: Successfully displayed fetched data in a grid layout on the front-end.



Final Outcome

- **Sanity CMS**: Populated with imported data.
- **Front-end**: Data displayed successfully in a user-friendly format.
- **Verification**: All components and workflows tested and validated.

Conclusion

Through a structured and systematic process, we successfully integrated external API data into Sanity Studio and displayed it seamlessly on the front-end of the Next.js project. The following milestones were achieved:

1. **Sanity Studio Setup:** Installed and configured Sanity Studio as a CMS for managing content, ensuring it integrates effectively with the Next.js project.
2. **Environment Variables:** Securely configured environment variables to manage sensitive information such as project ID, data set, and API token.
3. **Data Import:** Developed and executed a robust script to fetch API data, process it, and populate Sanity Studio with the required data fields.
4. **Frontend Integration:** Fetched data from Sanity Studio and displayed it dynamically in a visually appealing grid layout on the front-end using reusable components and modern design practices.
5. **Verification:** Ensured the accuracy and functionality of the workflow through thorough testing and debugging.

This workflow ensures a scalable, reusable, and efficient approach to integrating and managing external data sources in a modern web application, leveraging the capabilities of Sanity Studio and Next.js. The successful outcome highlights the effectiveness of the implementation and sets a strong foundation for future enhancements.