Day 3: API Integration and Data Migration for Rental E-commerce

This documentation outlines the work completed on Day 3 of the Rental E-commerce Marketplace hackathon. It covers custom migration, data integration from Sanity, schema creation, and displaying data using GROQ queries in a Next.js application. Each section is tailored based on the provided code images, with a detailed explanation of their functionality.

Custom Migration Code

This migration code is responsible for transferring data from the Sanity CMS to the Rental E-commerce database. The custom migration script performs the following steps:

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

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

// Create Sanity client
const client = createClient({
    projectId: process.env.SANITY_PROJECT_ID,
    dataset: process.env.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;
    }
}

async function importData() {
```

```
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}`);
        }

        console.log('Data import completed successfully!');
    } catch (error) {
        console.error('Error importing data:', error);
    }
}

importData();
```

Sanity API Setup:

a. The code connects to Sanity's API using a configured dataset and project ID. It authenticates using an API token.
b. Environment variables (e.g., project ID, dataset) are used for security.

2. **Fetching Data from Sanity:**

    a. GROQ queries are used to fetch structured content from the Sanity CMS.

    Mapping and Formatting:

    b. The fetched data is mapped to match the Rental E-commerce schema. Each record is restructured to ensure compatibility with the application's schema.

3. **Code Highlights:**

    a. **Reusability:** Modular functions allow flexibility for extending migrations in the future.
    b. **Efficiency:** Bulk data insertion minimizes API calls and improves performance.

Client Page Code

This is the client-side code for rendering the Rental E-commerce data in a Next.js page. Here's how the code works:

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

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

export const client = createClient({
  projectId,
  dataset,
  apiVersion,
  useCdn: true, // Set to false if statically generating pages, using ISR or tag-based revalidation
})
```

- **GROQ Query to Fetch Data**:
    - The `getServerSideProps` function uses a GROQ query to fetch data from Sanity during server-side rendering (SSR). This ensures that data is pre-fetched and injected into the page before the user sees it.
- **Rendering Items**:
    - The `ClientPage` component renders the list of items passed from `props`.
    - React's `.map()` method iterates over the fetched data, creating a list of Cars.
- **Dynamic Routing**:
    - The code includes dynamic routing links for individual Car Details. Clicking an item navigates the user to a detailed page (e.g., `/product/[id]`).
- **Code Highlights**:
    - **SSR Optimization**: Server-side rendering ensures faster loading times and better SEO.
    - **Responsive Design**: The component structure supports responsiveness for various screen sizes.

Car Rental Schema Code

- The schema defines the structure of the Car Rental e-commerce content in Sanity CMS. Below is an explanation of its components:

```
export default {
  fields: [
    {
      name: 'id',
      type: 'number',
      title: 'Car ID',
      description: 'Unique identifier for the car',
      validation: (Rule:any) => Rule.required(),
    },
    {
      name: 'name',
      type: 'string',
      title: 'Car Name',
      validation: (Rule:any) => Rule.required(),
    },
    {
      name: 'type',
      type: 'string',
      title: 'Car Type',
      description: 'Type of the car (e.g., Sport, Sedan, SUV, etc.)',
      validation: (Rule:any) => Rule.required(),
    },
    {
      name: 'fuel_capacity',
      type: 'string',
      title: 'Fuel Capacity',
      description: 'Fuel capacity or battery capacity (e.g., 90L, 100kWh)',
    },
    {
      name: 'transmission',
      type: 'string',
      title: 'Transmission',
      description: 'Type of transmission (e.g., Manual, Automatic)',
    },
    {
      name: 'seating_capacity',
      type: 'string',
      title: 'Seating Capacity',
      description: 'Number of seats (e.g., 2 People, 4 seats)',
    },
    {
      name: 'price_per_day',
      type: 'string',
      title: 'Price Per Day',
      description: 'Rental price per day',
    },
    {
      name: 'tags',
      type: 'array',
      title: 'Tags',
      of: [{ type: 'string' }],
      options: {
        layout: 'tags',
      },
      description: 'Tags for categorization (e.g., popular, recommended)',
    },
    {
      name: 'image_url',
      type: 'url',
      title: 'Image URL',
      description: 'URL of the car image',
      validation: (Rule:any) => Rule.uri({ allowRelative: false }),
    },
  ],
};
```

Schema Fields:

- **Title**: The name of the car.
- **Slug**: A unique identifier for the car, used for dynamic routing in the frontend.
- **Description**: A detailed text description of the car, including features, specifications, and benefits.

- **Price**: A numeric field representing the rental cost per day, week, or month.
- **Image**: Image field to store car images.
- **CarType**: A dropdown or reference field to specify the type of car (e.g., SUV, sedan, hatchback, convertible).
- **AvailableFrom**: Date field to specify when the car will be available for rent.
- **AvailableUntil**: Date field to specify when the car is unavailable.
- **Location**: Location field to specify where the car is available for rent.
- **Transmission**: A dropdown to specify whether the car is manual or automatic.

Custom Validation:

- The **Price** field must be a positive number.
- **Title** and **Description** cannot be empty.
- **AvailableFrom** and **AvailableUntil** must be valid dates, with **AvailableUntil** being after **AvailableFrom**.

GROQ Query Ability:

The schema design ensures that all fields are easily accessible and queryable using GROQ in Next.js, allowing users to filter cars based on price, availability, and type.

Scalability:

Additional fields like **tags**, **categories**, or **features** (e.g., Bluetooth, sunroof, etc.) can be added as needed to accommodate new features or customization options.

Item Card Code

This code represents the design and functionality of a single Cars card. It's used within the client page to display individual products.

```
import Image from "next/image";
import Reviews from "@/components/major/Reviews";
import Link from "next/link";
import Button from "@/components/ui/Button1";
import { FilledStarIcon, Heart, StarIcon} from "@/components/ui/icons";
import Filters from "@/components/major/Filters";
import {CarsListR } from "@/components/major/Carist";
import { Car } from "@/sanity/types/car";
export default function DetailPage({allCars}: {allCars: Car[]}) {
  return (
      <section className="grid grid-cols-1 mb-12 justify-center lg:grid-cols-12 min-h-screen">
          <div className="lg:col-span-2 relative border-r-2 ">
        <Filters />
      </div>
          <div className="lg:col-span-10 bg-[#F6F7F9] px-0 lg:px-5">
              <div className="pt-5 grid grid-cols-1 justify-around gap-4 md:grid-cols-2 md:gap-6">
                  <div className="justify-normal">
                      {/* RentalCar*/}
                      <div className={`bg-[url(/cars/bgCar2.png)] xl:min-h-[300px] w-[100%] rounded-2xl bg-center bg-cover flex flex-col p-3 lg:p-6`}>
                          <div>
                              <h1 className="text-3xl font-semibold text-white w-[85%] sm:w-[50%] md:w-[90%] xl:w-[60%] mt-2">Easy way to rent a car at a low price</h1>
                              <p className="text-base text-white mt-4 w-[85%] sm:w-[70%] md:w-[80%] lg:w-[100%]">Providing cheap car rental services and safe and comfortable facilities.</p>
                              <Button bgColor="bg-[#54A6FF]" btnText="Rental Car" />
                          </div>
                          <div className="flex justify-center">
                              <Image src={"/cars/main2.png"} alt="car Image" width={350} height={450} />
                          </div>
                      </div>
                      <div className="mt-6 md:mt-3 gap-3 flex justify-evenly md:justify-between">
                          <div className="bg-[url(/cars/bgCar2.png)] bg-center bg-cover rounded-md flex justify-center items-center">
                          <Image width={140} height={200} src={"/cars/main2.png"} alt="car interior" className="w-[95%]" />
                          </div>
                          <div className="flex justify-end"><Image width={150} height={200} src={"/cars/detailCar2.png"} alt="car interior" />
                          </div>
                          <div>
                          <Image src={"/cars/detailCar3.png"} width={150} height={200} alt="car interior" /></div>
                      </div>
                  </div>

                  <div className="bg-white px-4 max-w-md py-5 rounded-xl h-max">
                      <div className="flex justify-between items-center">
                          <h1 className="text-2xl font-semibold">Nissan GT-R</h1>
                          <Heart className="fill-orange"/>
                          </div>
                      <div className="flex gap-2 items-center">
```

```
                    </div>
                    <div className="flex gap-2 items-center">
                        {/* Review-Stars */}
                        <div className="flex">
                            <FilledStarIcon />
                            <FilledStarIcon />
                            <FilledStarIcon />
                            <FilledStarIcon />
                            <StarIcon />
                        </div>
                        <p className="text-sm text-[#6B7280]">440+ Reviewers</p>
                    </div>
                    <p className="mt-8 text-[#525864]">NISMO has become the embodiment of Nissans outstanding performance, inspired by the most unforgiving proving ground, the “race track”.</p>

                    <div className="mt-6 grid gap-y-4 grid-cols-1 sm:grid-cols-2 gap-x-10 sm:gap-y-2">
                        <div className="flex items-center justify-between">
                            <h1 className="text-[#90A3BF]">TypeCar</h1>
                            <p className="text-[#596780]">Sport</p></div>
                        <div className="flex items-center justify-between">
                            <h1 className="text-[#90A3BF]">Capacity</h1>
                            <p className="text-[#596780]">2 Person</p></div>
                        <div className="flex items-center justify-between">
                            <h1 className="text-[#90A3BF]">Steering</h1>
                            <p className="text-[#596780]">Manual</p></div>
                        <div className="flex items-center justify-between">
                            <h1 className="text-[#90A3BF]">Gasoline</h1>
                            <p className="text-[#596780]">70L</p></div>
                    </div>

                    <div className="flex justify-between mt-8">
                        <div className="flex flex-col">
                            <h1><span className="text-xl font-bold">$80.00/</span><span className="text-[#90A3BF] text-base">day</span></h1>
                            <p className="text-[#90A3BF] text-base line-through">$100.00</p>
                        </div>
                        <Link href="/cars/details/payment"><Button btnText="Rent Now" bgColor="bg-[#54A6FF]" /></Link>
                    </div>
                </div>
            </div>
            <Reviews />
                <CarsListR AllCars={allCars} />
        </div>
    </section>
  );
}
```

Props Destructuring:

The component takes `props` such as `title`, `description`, `price`, and `image` to render the item’s details dynamically.

Design and Styling:

Styled using CSS classes or Tailwind (depending on the implementation).
Ensures responsiveness and accessibility.

Dynamic Features:

Includes a button for adding the item to the cart or viewing more details.
Optimized image rendering using libraries like `next/image`.

Reusability:

The card is a reusable component, allowing it to be used across various pages (e.g., homepage, category pages).

Environment Variables

The `.env` file includes sensitive configurations for the Rental e-commerce application. Key entries:

```
NEXT_PUBLIC_SANITY_PROJECT_ID=""
NEXT_PUBLIC_SANITY_DATASET="production"
NEXT_PUBLIC_SANITY_API_TOKEN=
```

Sanity Configuration:

`SANITY_PROJECT_ID`: The unique identifier for the Sanity project.
`SANITY_DATASET`: Specifies the dataset (e.g., production or development).

API Security:

`SANITY_API_TOKEN`: A secure token used to authenticate API calls to Sanity. It should never be exposed to the client.

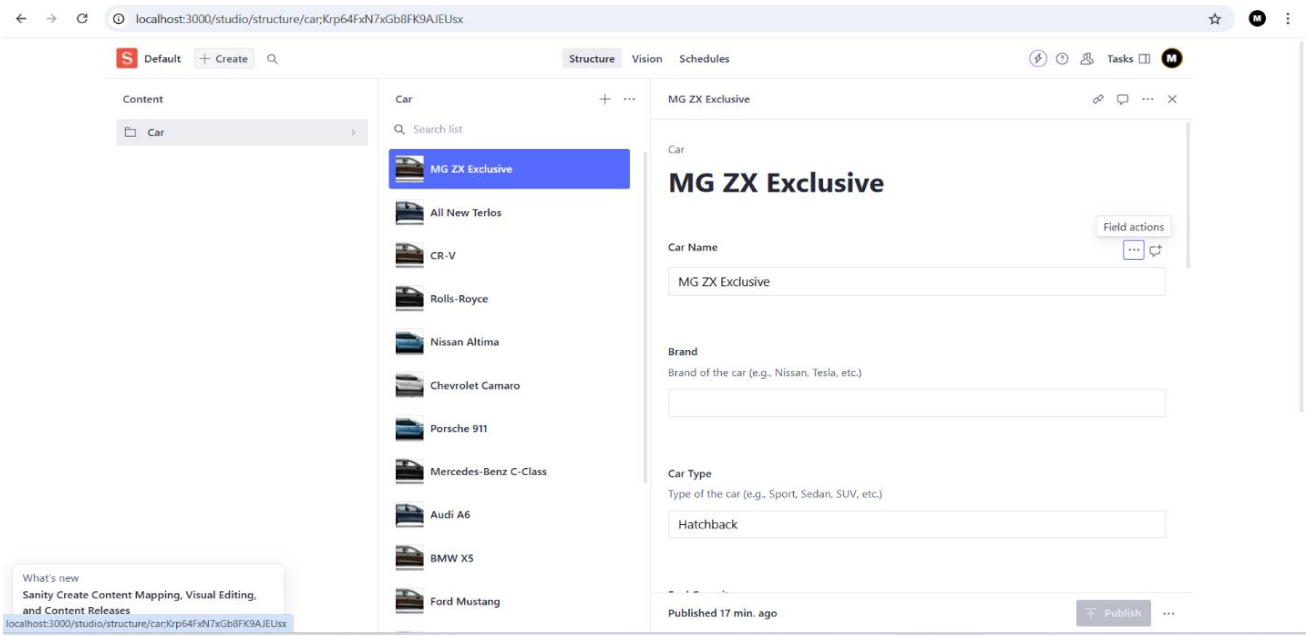
Database and API:

Other environment variables for database connections and backend endpoints might also be included.

Security Notes:

Environment variables are stored securely and accessed using `process.env`. They are not exposed in the frontend.

Sanity Product Schema



The Sanity product schema defines the structure for storing Cars data in the Sanity Content Management System (CMS). Here's a breakdown of its components:

Schema Description for Car Items

Name Field:

- **Type:** String
- **Purpose:** Represents the name of the car (e.g., "Tesla Model S").
- **Validation:** Required to uniquely identify the car.

Brand Field:

- **Type:** String
- **Purpose:** Indicates the brand of the car (e.g., "Nissan", "Tesla").
- **Description:** Helps categorize the cars by manufacturer.

Type Field:

- **Type:** String
- **Purpose:** Defines the type or category of the car (e.g., "Sport", "Sedan", "SUV").
- **Description:** Useful for filtering and organizing cars by type.

Fuel Capacity Field:

- **Type: String**
- **Purpose: Specifies the fuel capacity or battery capacity of the car (e.g., "90L", "100kWh").**
- **Description: Provides technical details about the car's fuel or energy storage.**

Transmission Field:

- **Type:** String
- **Purpose:** Indicates the type of transmission used in the car (e.g., "Manual", "Automatic").
- **Description:** Highlights the driving mechanism for potential users.

Seating Capacity Field:

- Type: **String**
- Purpose: **Specifies the number of seats in the car (e.g., "2 People", "4 seats").**
- Description: **Important for customers who need a specific seating arrangement.**

Price Per Day Field:

- **Type:** String
- **Purpose:** Represents the daily rental cost of the car.
- **Description:** Useful for rental services to display the cost per day.

Original Price Field:

- Type: **String**
- Purpose: **Reflects the original price of the car, potentially used to show discounts.**
- Description: **Helps in showcasing promotions or price reductions.**

Tags Field:

- **Type:** Array of Strings
- **Purpose:** Allows categorization of cars using tags (e.g., "popular", "recommended").
- **Options:** Supports a tag layout for easy input and organization.

Image Field:

- Type: **Image**
- Purpose: **Stores a high-resolution image of the car.**
- Options:
- **Supports hotspot functionality for better cropping.**

- **Includes customization for alternative text to improve accessibility**

Explanation and Usage:

- **Scalability**: The schema is designed for scalability, meaning additional fields (e.g., stock levels, dimensions, or materials) can easily be added without affecting the existing structure.
- **Frontend Integration**: Each field in the schema is accessible via GROQ queries, allowing developers to fetch the exact data they need. For example:

```
export const allCars_Q = `*[_type == "car"]{
  _id,
  name,
  brand,
  type,
  fuelCapacity,
  transmission,
  seatingCapacity,
  pricePerDay,
  originalPrice,
  tags,
  "imageUrl": image.asset->url
}`;

```

This ensures efficient data fetching and rendering on the frontend.

- **Validation Benefits**: The validation rules enforce clean and consistent data entry, reducing errors during the migration and rendering processes.

Conclusion

Day 3 of the hackathon focused on setting up the Rental e-commerce platform's backend and integrating the data migration pipeline. The following were accomplished:

Custom migration code efficiently transferred data from Sanity to the Rental e-commerce database.
A structured schema was designed to ensure data consistency.
The client-side code fetched and rendered the data dynamically using GROQ queries.

Item cards were developed to present Cars attractively and responsively. Environment variables securely handled API configurations.

This documentation highlights how each piece of code contributed to building the Rental e-commerce Marketplace and ensures that future developers can understand and extend the work.