Marketplace Technical Foundation: Rental E-Commerce

Introduction

This document outlines the technical foundation for the development of a Rental E-Commerce marketplace. The goal is to ensure a robust, scalable, and efficient platform that aligns with the business requirements and user needs defined during the initial planning phase.

---

1. Technical Requirements

Frontend Requirements

- **User-Friendly Interface**: Intuitive navigation and product browsing.
- **Responsive Design**: Fully optimized for mobile and desktop users.
- **Pages to Develop**:
  1. **Home**: Highlight featured products and promotions.
  2. **Product Listing**: Display products with filtering and sorting options.
  3. **Product Details**: Detailed view of a selected product with pricing and availability.
  4. **Cart**: Display selected items, total cost, and an option to proceed to checkout.
  5. **Checkout**: Secure payment page with user details.
  6. **Order Confirmation**: Summary of the completed order with tracking information.

Backend Requirements (Sanity CMS)

- **Product Management**:
  - Schema for storing product details: Name, Description, Price, Rental Duration, Stock Availability, and Images.
- **Customer Management**:
  - Schema for storing user details: Name, Contact Information, and Order History.
- **Order Management**:
  - Schema for tracking orders: Customer Info, Product Details, Payment Status, and Delivery Status.

Third-Party APIs

- **Payment Gateway**:
  - Integrate Stripe or PayPal for secure and efficient payment processing.
- **Shipment Tracking**:

    - Use APIs like AfterShip or Shippo to provide real-time tracking information to customers.

---

2. API Integration Process

Step 1: Identify Required APIs

- **Payment Gateway API**:
    - Choose a provider (e.g., Stripe, PayPal) based on transaction volume and region.
- **Shipment Tracking API**:
    - Use solutions like AfterShip or Shippo for real-time delivery updates.

Step 2: Obtain API Keys

- Sign up for developer accounts on the chosen platforms.
- Retrieve API keys and configure them in a secure environment (e.g., environment variables).

Step 3: Integrate APIs in Backend

- **Payment API**:
    - Implement API calls for creating transactions, handling errors, and processing refunds.
    - Ensure secure handling of sensitive data like credit card details using encryption.
- **Shipment API**:
    - Integrate endpoints for creating shipment orders and retrieving tracking details.

Step 4: Connect Frontend to API

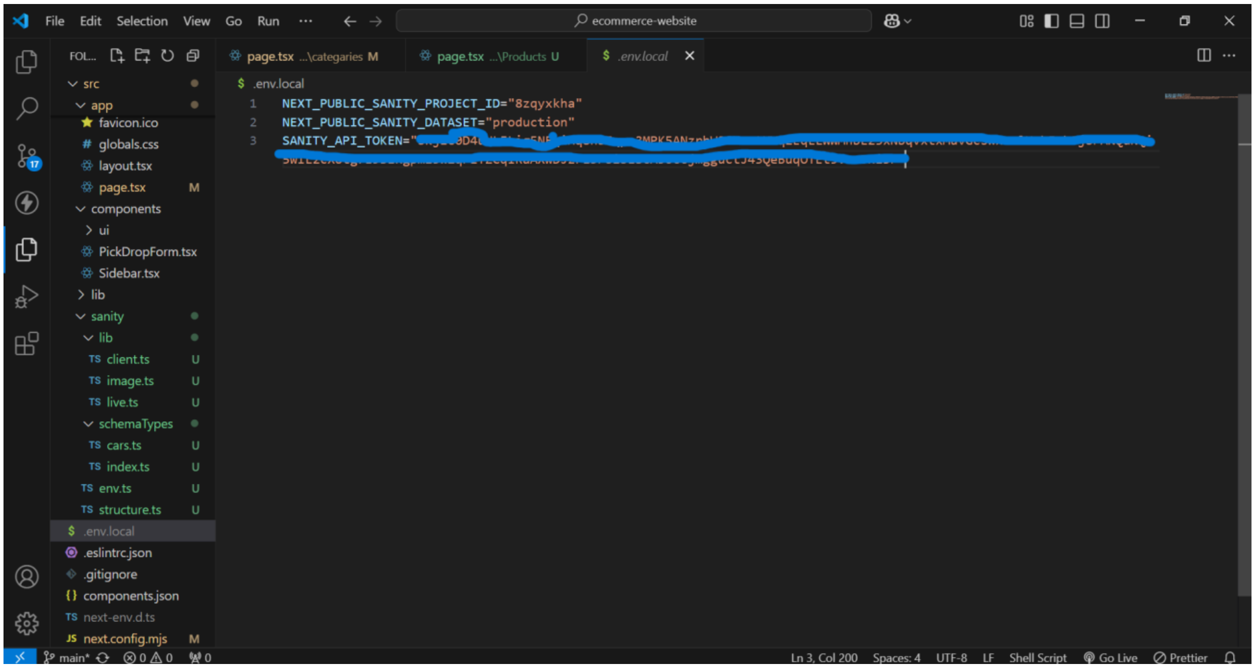
- Use API endpoints in the frontend to send user data (e.g., payment info, order details) to the backend.
- Display real-time information like shipment status and payment confirmation.

Step 5: Test API Integration

- Use sandbox environments provided by APIs for testing.
- Validate responses and error handling for various scenarios (e.g., failed payments, invalid tracking numbers).

Step 6: Monitor and Optimize

- Use monitoring tools to track API performance.
- Optimize API calls to reduce latency and improve user experience.

- 

```
NEXT_PUBLIC_SANITY_PROJECT_ID="8zqyxkha"
NEXT_PUBLIC_SANITY_DATASET="production"
SANITY_API_TOKEN="[redacted]
```

---

3. Adjustments Made to Schemas

Product Schema

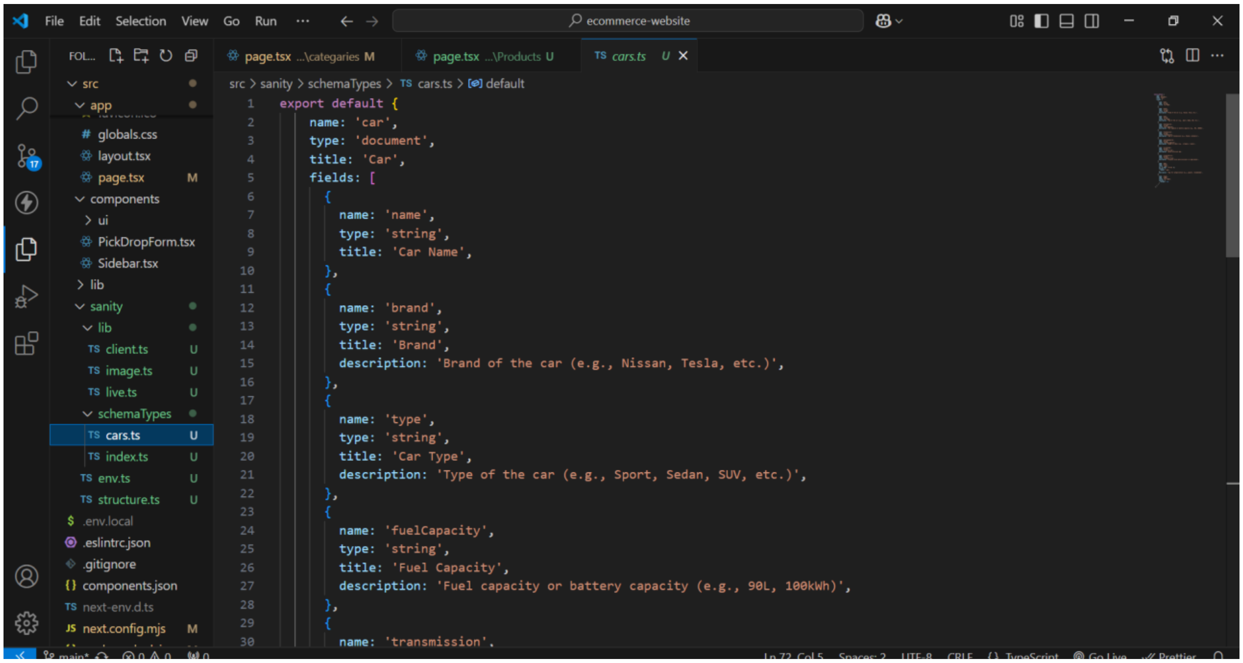
- **Added Fields**:
  - Rental Duration: Number field to specify the rental period in days.
  - Availability Status: Boolean field to indicate if a product is currently available for rent.

Order Schema

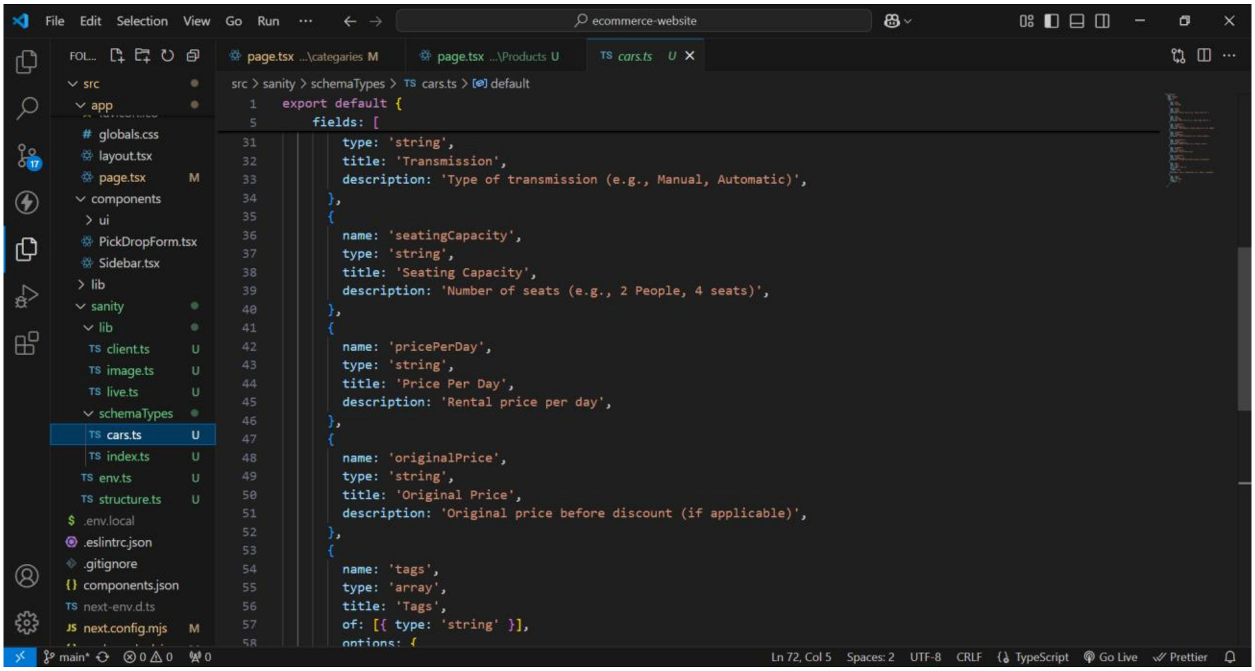
- **Added Fields**:
  - Rental Start Date: Date field to track when the rental begins.
  - Rental End Date: Date field to track when the rental ends.
  - Total Rental Cost: Number field calculated based on price and rental duration.

Customer Schema

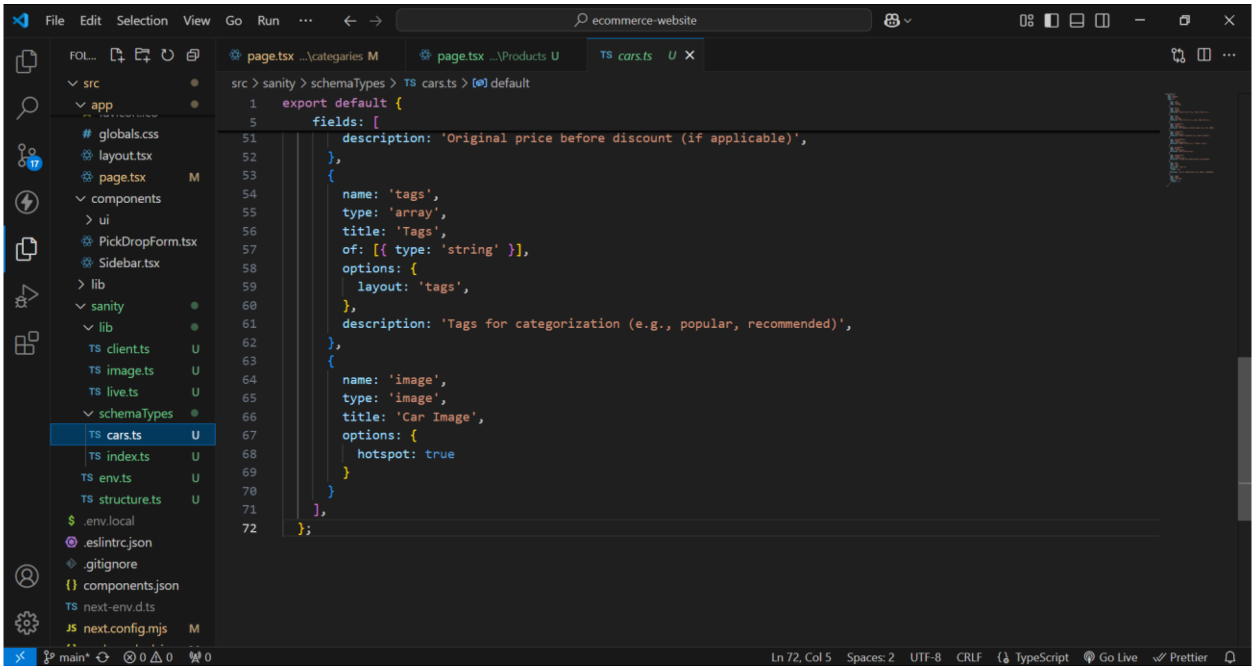
- **Added Fields**:
  - Address: Text field for storing customer delivery address.
  - Payment Method: String field to track the chosen payment method (e.g., Credit Card, PayPal).

- 

```
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',
```



```
export default {
  fields: [
      type: 'string',
      title: 'Transmission',
      description: 'Type of transmission (e.g., Manual, Automatic)',
    },
    {
      name: 'seatingCapacity',
      type: 'string',
      title: 'Seating Capacity',
      description: 'Number of seats (e.g., 2 People, 4 seats)',
    },
    {
      name: 'pricePerDay',
      type: 'string',
      title: 'Price Per Day',
      description: 'Rental price per day',
    },
    {
      name: 'originalPrice',
      type: 'string',
      title: 'Original Price',
      description: 'Original price before discount (if applicable)',
    },
    {
      name: 'tags',
      type: 'array',
      title: 'Tags',
      of: [{ type: 'string' }],
      options: {
```



```
export default {
  fields: [
      description: 'Original price before discount (if applicable)',
    },
    {
      name: 'tags',
      type: 'array',
      title: 'Tags',
      of: [{ type: 'string' }],
      options: {
        layout: 'tags',
      },
      description: 'Tags for categorization (e.g., popular, recommended)',
    },
    {
      name: 'image',
      type: 'image',
      title: 'Car Image',
      options: {
        hotspot: true
      }
    }
  ],
};
```

4. Migration Steps and Tools Used

Migration Overview

Data migration ensures that existing data is aligned with the updated schemas without loss or inconsistency.

Step 1: Backup Existing Data

- Export current datasets from Sanity CMS using the `sanity dataset export` command.
- Store the backup securely.

Step 2: Update Schemas

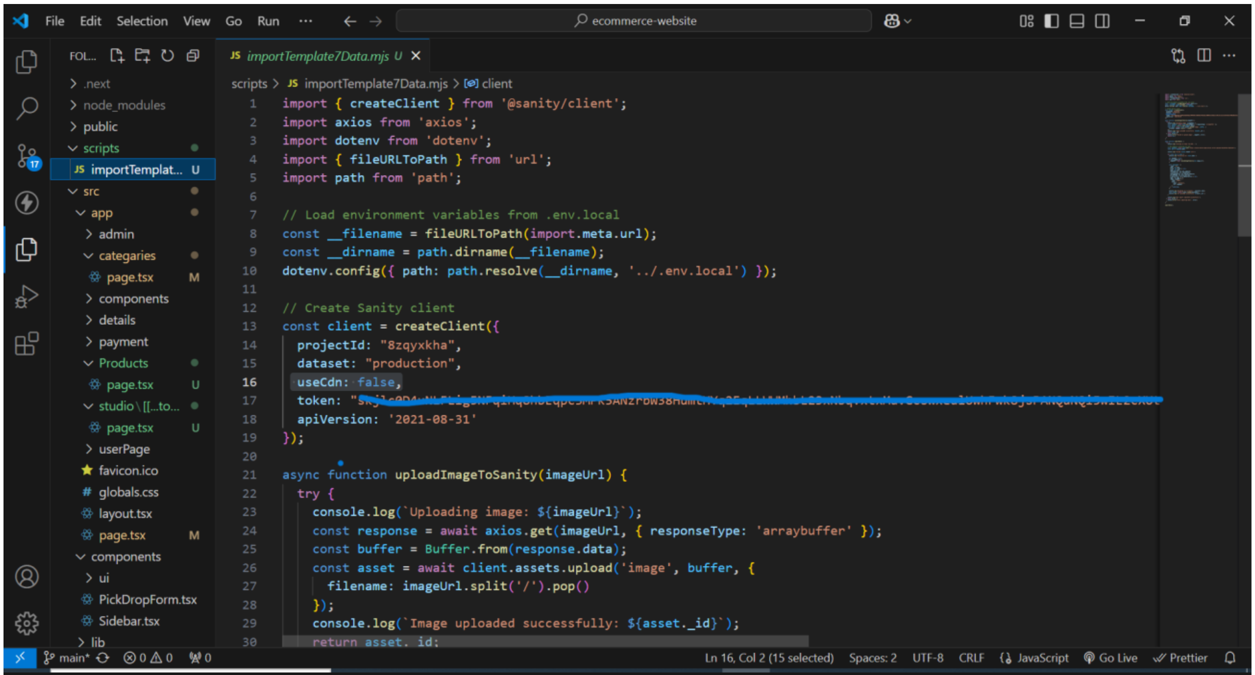
- Modify the schema files in the Sanity CMS project to reflect the new fields and changes.

Step 3: Deploy Schema Updates

- Deploy updated schemas using the `sanity deploy` command.
- Validate the deployment in the CMS Studio.
-

Step 4: Transform Data

- Use migration scripts to adjust existing data to match the new schema format.
- Example: Add default values for new fields like `availabilityStatus`.

- 

scripts > importTemplate7Data.mjs > client

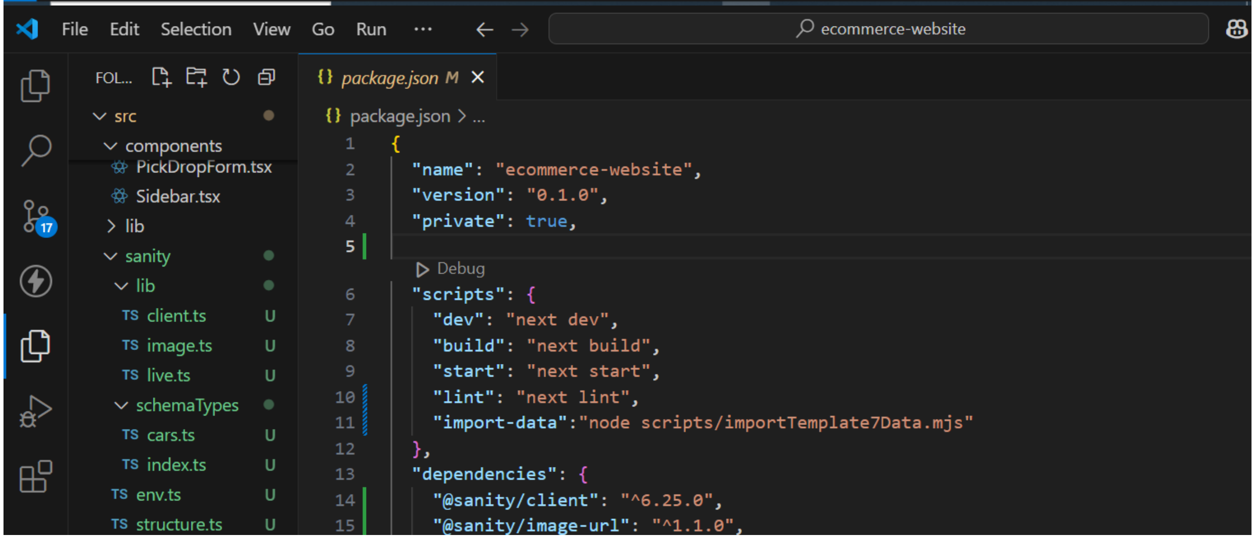
```javascript
import { createClient } from '@sanity/client';
import axios from 'axios';
import dotenv from 'dotenv';
import { fileURLToPath } from 'url';
import path from 'path';

// 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: "8zqyxkha",
  dataset: "production",
  useCdn: false,
  token: "[redacted]
  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;
```

Ln 16, Col 2 (15 selected) Spaces: 2 UTF-8 CRLF JavaScript Go Live Prettier



package.json > ...

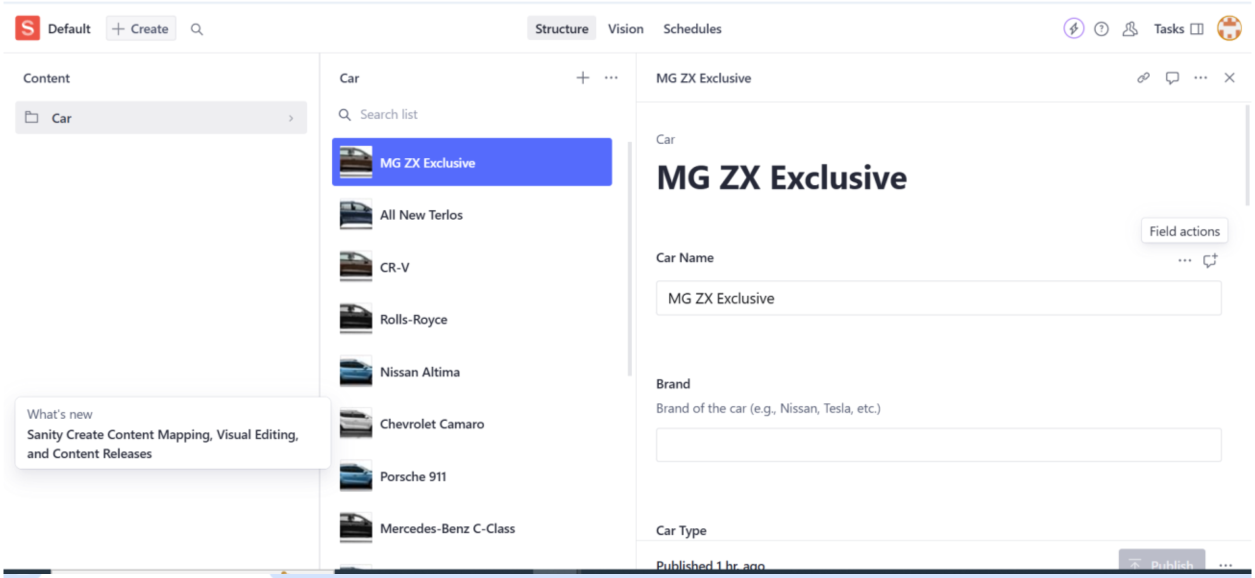
```json
{
  "name": "ecommerce-website",
  "version": "0.1.0",
  "private": true,
  ▷ Debug
  "scripts": {
    "dev": "next dev",
    "build": "next build",
    "start": "next start",
    "lint": "next lint",
    "import-data":"node scripts/importTemplate7Data.mjs"
  },
  "dependencies": {
    "@sanity/client": "^6.25.0",
    "@sanity/image-url": "^1.1.0",
```

Step 5: Import Transformed Data

- Use the `sanity dataset import` command to re-import the updated datasets.
- Verify data integrity post-import.

Tools Used

- **Sanity CLI**: For exporting, importing, and deploying schemas.
- **Node.js Scripts**: For transforming data during migration.
- **Postman**: For testing API endpoints during data updates.



MG ZX Exclusive
All New Terlos
CR-V
Rolls-Royce
Nissan Altima
Chevrolet Camaro
Porsche 911
Mercedes-Benz C-Class

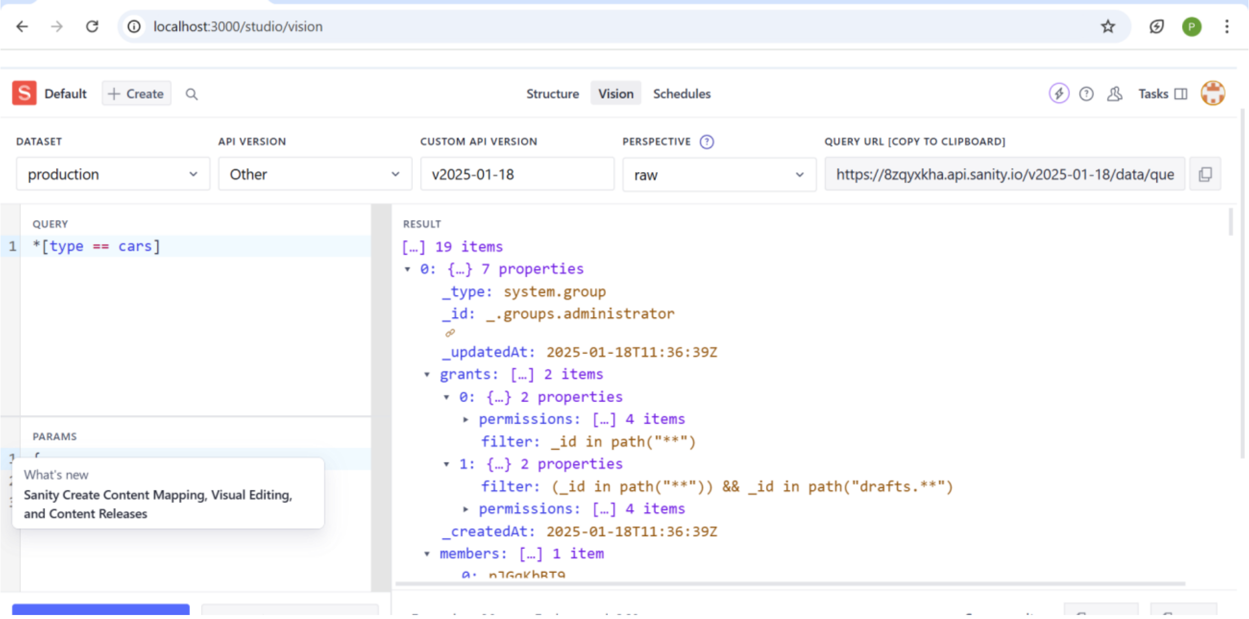
Car

MG ZX Exclusive

Car Name: MG ZX Exclusive

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

Car Type


localhost:3000/studio/vision
Default
Create
Structure
Vision
Schedules
Tasks
What's new
Sanity Create Content Mapping, Visual Editing, and Content Releases

DATASET: production
API VERSION: Other
CUSTOM API VERSION: v2025-01-18
PERSPECTIVE: raw
QUERY URL [COPY TO CLIPBOARD]: https://8zqyxkha.api.sanity.io/v2025-01-18/data/que

QUERY

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

PARAMS

RESULT

```
[…] 19 items
▾ 0: {…} 7 properties
    _type: system.group
    _id: _.groups.administrator
    _updatedAt: 2025-01-18T11:36:39Z
  ▾ grants: […] 2 items
    ▾ 0: {…} 2 properties
      ▸ permissions: […] 4 items
        filter: _id in path("**")
    ▾ 1: {…} 2 properties
        filter: (_id in path("**")) && _id in path("drafts.**")
      ▸ permissions: […] 4 items
    _createdAt: 2025-01-18T11:36:39Z
  ▾ members: […] 1 item
```

5. Conclusion

This technical foundation provides a clear roadmap for building a Rental E-Commerce marketplace. By adhering to this plan, the platform will align with business goals, offer a seamless user experience, and ensure scalability for future growth.