Day 4: Building Dynamic Front_end Components for Your Marketplace

---

1. Introduction

On Day 4 of the project, the focus was on building dynamic front_end components that interact seamlessly with the back_end. This was achieved by integrating the Sanity CMS to manage dynamic data (such as car listings and booking availability) and incorporating it into the marketplace's front_end using React. The key objective was to ensure the marketplace adapts to changing data in real time, providing an interactive and dynamic user experience.

---

2. Sanity CMS Setup and Integration

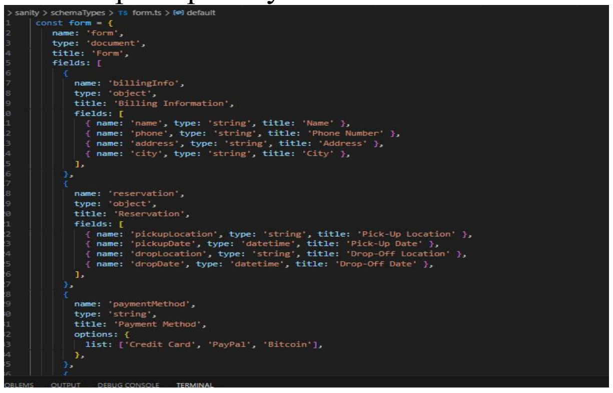
To manage dynamic content for the marketplace, the project utilized **Sanity CMS** as the content management system. Sanity provides a flexible, real-time content platform, which was leveraged to create dynamic car rental listings, availability schedules, and prices. The main steps in setting up Sanity for the marketplace were as follows:

2.1. Sanity Schema Design

Sanity schemas define the structure of the content stored within the CMS. For this project, schemas were created for various dynamic content types, including:

Car Listings: The schema for car listings includes fields such as:

-
  - **Model**: Name of the car (e.g., "Toyota Corolla").
  - **Price**: Rental price per day.



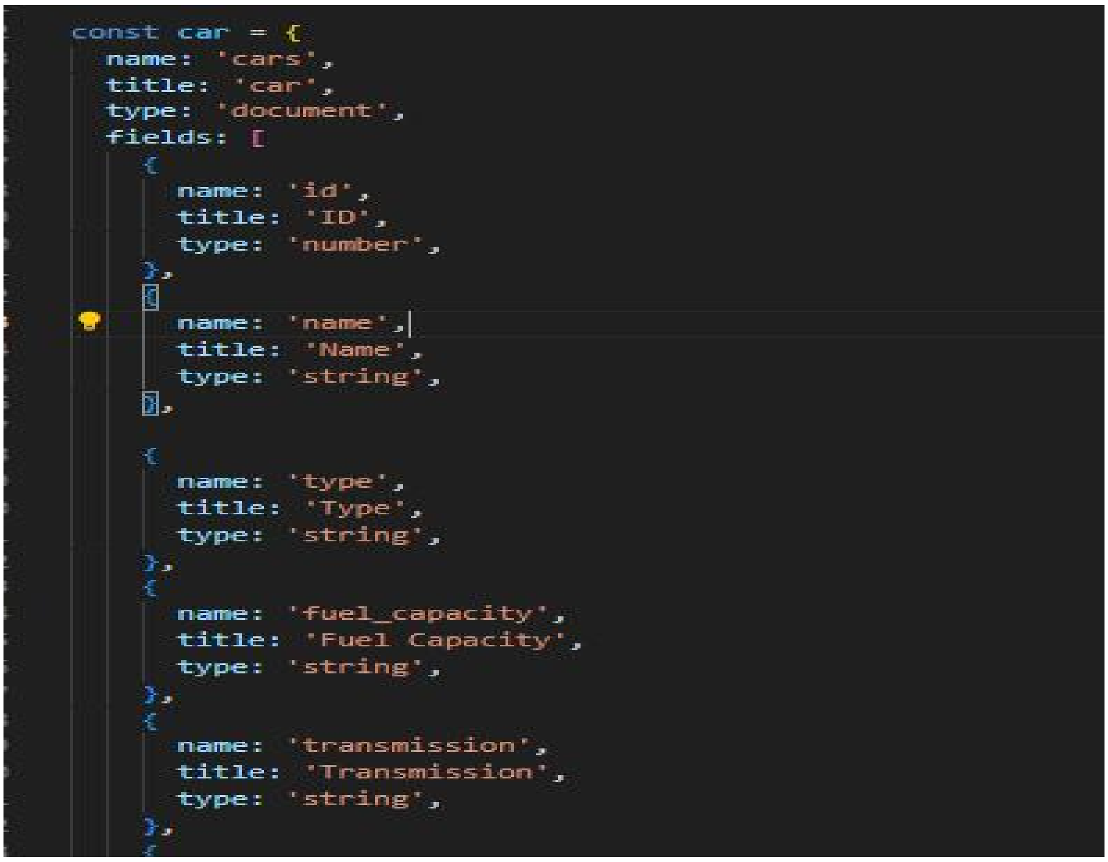
```
> sanity > schemaTypes > TS form.ts > [@] default
const form = {
    name: 'form',
    type: 'document',
    title: 'Form',
    fields: [
      {
        name: 'billingInfo',
        type: 'object',
        title: 'Billing Information',
        fields: [
          { name: 'name', type: 'string', title: 'Name' },
          { name: 'phone', type: 'string', title: 'Phone Number' },
          { name: 'address', type: 'string', title: 'Address' },
          { name: 'city', type: 'string', title: 'City' },
        ],
      },
      {
        name: 'reservation',
        type: 'object',
        title: 'Reservation',
        fields: [
          { name: 'pickupLocation', type: 'string', title: 'Pick-Up Location' },
          { name: 'pickupDate', type: 'datetime', title: 'Pick-Up Date' },
          { name: 'dropLocation', type: 'string', title: 'Drop-Off Location' },
          { name: 'dropDate', type: 'datetime', title: 'Drop-Off Date' },
        ],
      },
      {
        name: 'paymentMethod',
        type: 'string',
        title: 'Payment Method',
        options: {
          list: ['Credit Card', 'PayPal', 'Bitcoin'],
        },
      },
```

  - **Location**: Location where the car is available for rent.
  - **Availability**: Indicates if the car is available for the selected rental dates.
-

Booking Information: The schema for booking information includes fields for customer details, selected car, pick-up/drop-off locations, and rental dates.

-

By creating structured schemas, we ensured that adding or updating data in Sanity would reflect automatically in the marketplace's frontend.

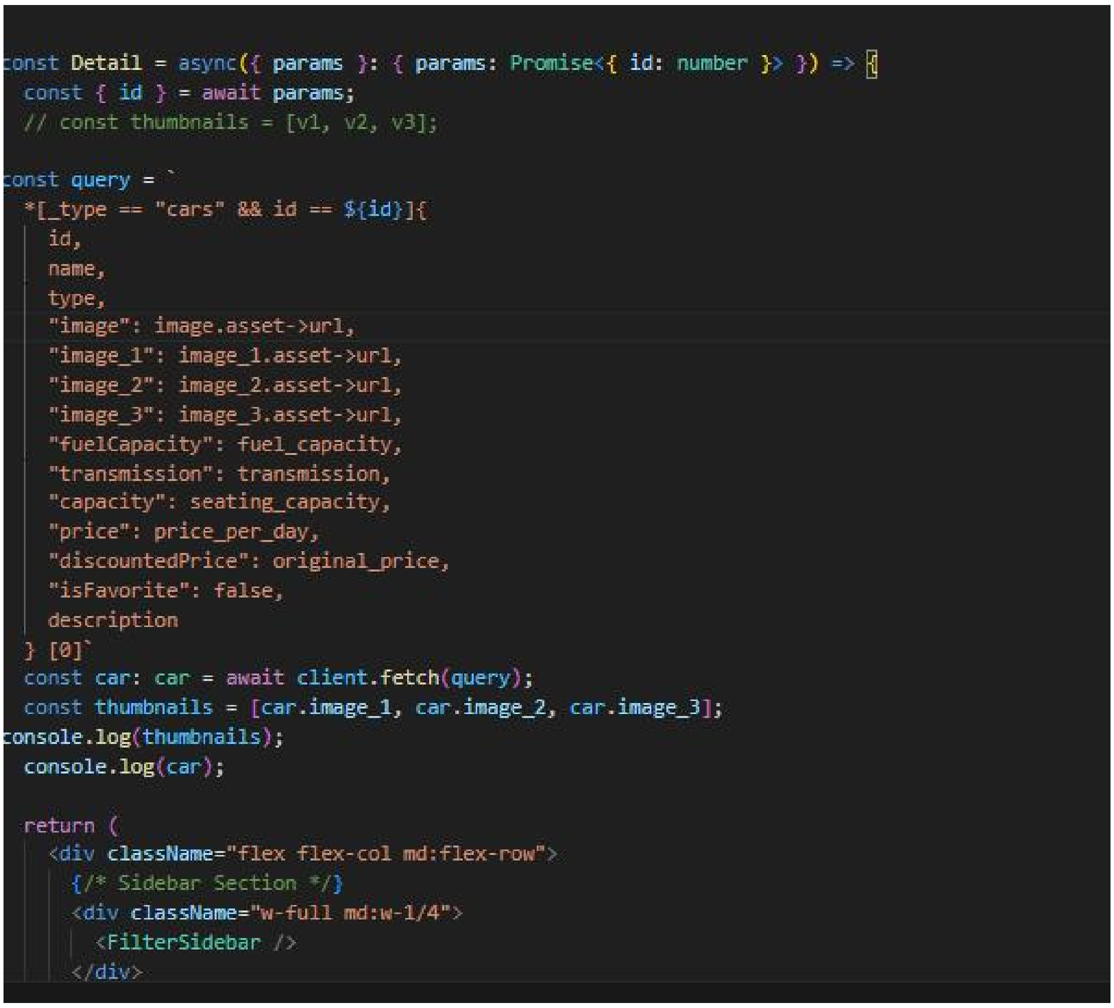


```
const car = {
  name: 'cars',
  title: 'car',
  type: 'document',
  fields: [
    {
      name: 'id',
      title: 'ID',
      type: 'number',
    },
    {
      name: 'name',
      title: 'Name',
      type: 'string',
    },

    {
      name: 'type',
      title: 'Type',
      type: 'string',
    },
    {
      name: 'fuel_capacity',
      title: 'Fuel Capacity',
      type: 'string',
    },
    {
      name: 'transmission',
      title: 'Transmission',
      type: 'string',
    },
```

2.2. Dynamic Content Fetching

The next step was fetching dynamic data from Sanity into the application. This was achieved using **GROQ (Graph-Relational Object Queries)**, which is Sanity's query language, to retrieve the latest content.


```
onst Detail = async({ params }: { params: Promise<{ id: number }> }) => {
  const { id } = await params;
  // const thumbnails = [v1, v2, v3];

onst query = `
  *[_type == "cars" && id == ${id}]{
    id,
    name,
    type,
    "image": image.asset->url,
    "image_1": image_1.asset->url,
    "image_2": image_2.asset->url,
    "image_3": image_3.asset->url,
    "fuelCapacity": fuel_capacity,
    "transmission": transmission,
    "capacity": seating_capacity,
    "price": price_per_day,
    "discountedPrice": original_price,
    "isFavorite": false,
    description
  } [0]`
  const car: car = await client.fetch(query);
  const thumbnails = [car.image_1, car.image_2, car.image_3];
onsole.log(thumbnails);
  console.log(car);

  return (
    <div className="flex flex-col md:flex-row">
      {/* Sidebar Section */}
      <div className="w-full md:w-1/4">
        <FilterSidebar />
      </div>
```

Example of a GROQ query for fetching car listings:

javascript
CopyEdit
```
const query = `*[_type == "car"]{
  model,
  price,
  location,
  availability
}`;
```

This query fetches all car listings from Sanity that match the "car" type, including details like model, price, and availability.

---

3. React Component Development for Dynamic UI

The dynamic data fetched from Sanity is used to build interactive UI components in React. React's **state management** and **props system** allow the UI to be updated automatically when the data changes.

3.1. Component Structure

The main React components for this dynamic marketplace include:

- **Car Listing Component**: This component displays all available cars. It fetches dynamic car data from Sanity and maps over the data to render individual car cards.
- **Car Detail Component**: This component displays detailed information about a selected car. It includes a booking form that allows users to select dates and finalize their rental.
- **Booking Form Component**: Handles input fields for the rental dates and personal information, dynamically interacting with the Sanity data to check car availability and process bookings.

3.2. Component Example: Car Listings

The **Car Listings** component displays a list of available cars, including details like the car model, price, and availability.

```
javascript
CopyEdit
const CarListing = ({ cars }) => {
  return (
    <div className="car-listing">
      {cars.map((car) => (
        <div className="car-card" key={car._id}>
          <h3>{car.model}</h3>
          <p>{car.price} per day</p>
          <p>{car.location}</p>
          <p>{car.availability ? "Available" : "Not Available"}</p>
        </div>
      ))}
    </div>
  );
};
```

This component takes in a list of cars (retrieved from Sanity) and maps over it to create individual car cards.

---

4. Dynamic Updates and State Management

To ensure that the frontend updates dynamically when data changes, **React's state management** is used extensively:

- **State Variables**: React state variables store dynamic content like car listings, booking details, and selected car.
- **UseEffect for Fetching Data**: The useEffect hook is employed to fetch the latest car listings from Sanity whenever the component is rendered.

Example of using useEffect to fetch data:

```
javascript
CopyEdit
import { useEffect, useState } from 'react';
const [cars, setCars] = useState([]);
useEffect(() => {
  fetchCarData();
}, []);
const fetchCarData = async () => {
  const res = await fetch('/api/cars');
  const data = await res.json();
  setCars(data);
};
```

This ensures that as soon as the component mounts, it fetches the most up-to-date data from the backend.

---

5. Error Handling and Data Validation

As dynamic data is fetched and rendered on the frontend, proper error handling and validation mechanisms are essential to ensure smooth user experience:

5.1. Data Fetching Errors

When fetching data from Sanity or other APIs, errors such as network failures or missing data can occur. These errors are handled by using try-catch blocks and providing user-friendly messages when things go wrong.

Example of error handling in data fetching:

```
javascript
CopyEdit
try {
  const data = await fetchCarData();
  if (!data) throw new Error('No cars available');
} catch (error) {
  console.error('Error fetching car data:', error);
  setError('Unable to load car listings. Please try again later.');
}
```

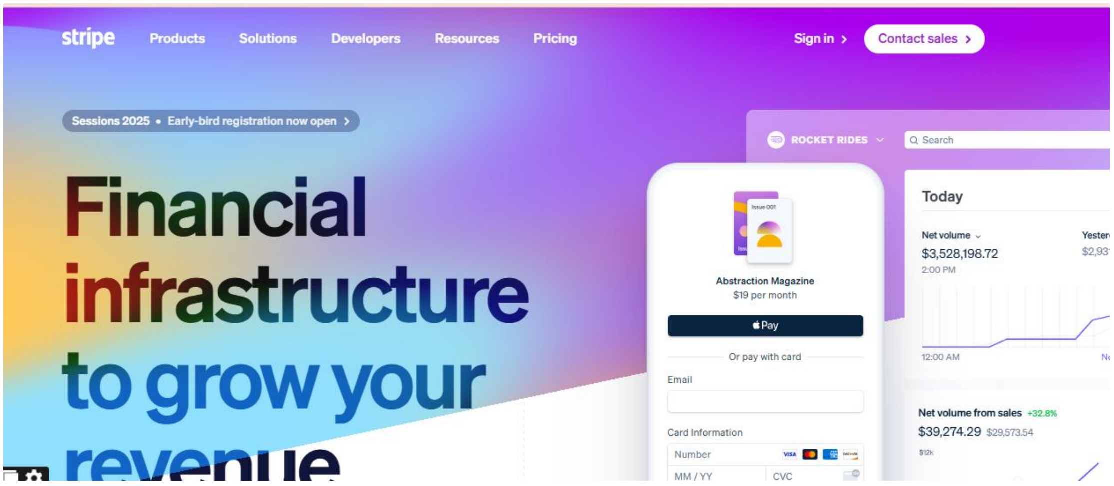
5.2. Form Validation

Form validation is crucial for ensuring that the data entered by users (such as rental dates or personal information) is correct. The form components use validation libraries (like Formik or React Hook Form) to ensure inputs are validated before submission.

6. Integrating Payment with Stripe

On Day 4, we also refined the integration with **Stripe** for processing payments:

- **Stripe Integration**: The Stripe payment system was integrated to handle payments for car rentals. The payment process checks if the user has provided valid payment details before proceeding with the booking.
- **Real-Time Payment Processing**: Payment confirmation data is linked with car availability in real-time. If a booking fails to process (e.g., payment not authorized), an error message is displayed.



7. Conclusion

Day 4 was focused on enhancing the frontend by making it dynamic and capable of handling real-time data from Sanity CMS. By integrating dynamic content fetching, React state management, and creating reusable components, the project now provides a seamless user experience. Additionally, by using Sanity for backend data management and integrating Stripe for payments, the marketplace is both scalable and flexible.