Day 5: Testing, Error Handling, and Back end Integration Refinement - Professional Documentation

---

1. Objective

Day 5's focus was on enhancing the back end integration of the car rental platform by refining dynamic data handling, implementing robust error handling mechanisms, and integrating Stripe for payment processing. Testing procedures were also conducted to ensure the platform's stability and user experience.

---

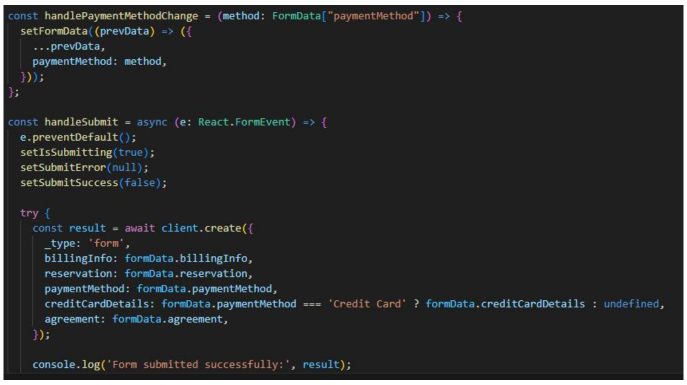
2. Testing

- **Unit Testing**: Verified back end functionalities like car availability, booking creation, and payment validation.
- **Integration Testing**: Ensured proper interaction between the front end (UI) and back end (data/API interactions).
- **Payment System Testing**: Tested Stripe integration, including payment processing and error handling for failed transactions.

---

3. Error Handling Implementation

- **Frontend Error Handling**:
  - Users are shown informative error messages when an API call fails.
  - Form validation prevents submission when required fields are left empty.
- **Backend Error Handling**:
  - **400**: Invalid data, e.g., missing fields.
  - **404**: Resource not found, e.g., car not available.
  - **500**: Internal server errors, with detailed logs for troubleshooting.
- **Graceful Degradation:**



```
const handlePaymentMethodChange = (method: FormData["paymentMethod"]) => {
  setFormData((prevData) => ({
    ...prevData,
    paymentMethod: method,
  }));
};

const handleSubmit = async (e: React.FormEvent) => {
  e.preventDefault();
  setIsSubmitting(true);
  setSubmitError(null);
  setSubmitSuccess(false);

  try {
    const result = await client.create({
      _type: 'form',
      billingInfo: formData.billingInfo,
      reservation: formData.reservation,
      paymentMethod: formData.paymentMethod,
      creditCardDetails: formData.paymentMethod === 'Credit Card' ? formData.creditCardDetails : undefined,
      agreement: formData.agreement,
    });

    console.log('Form submitted successfully:', result);
```

- The platform degrades gracefully by showing default information and allowing users to continue without major disruptions when partial failures occur.

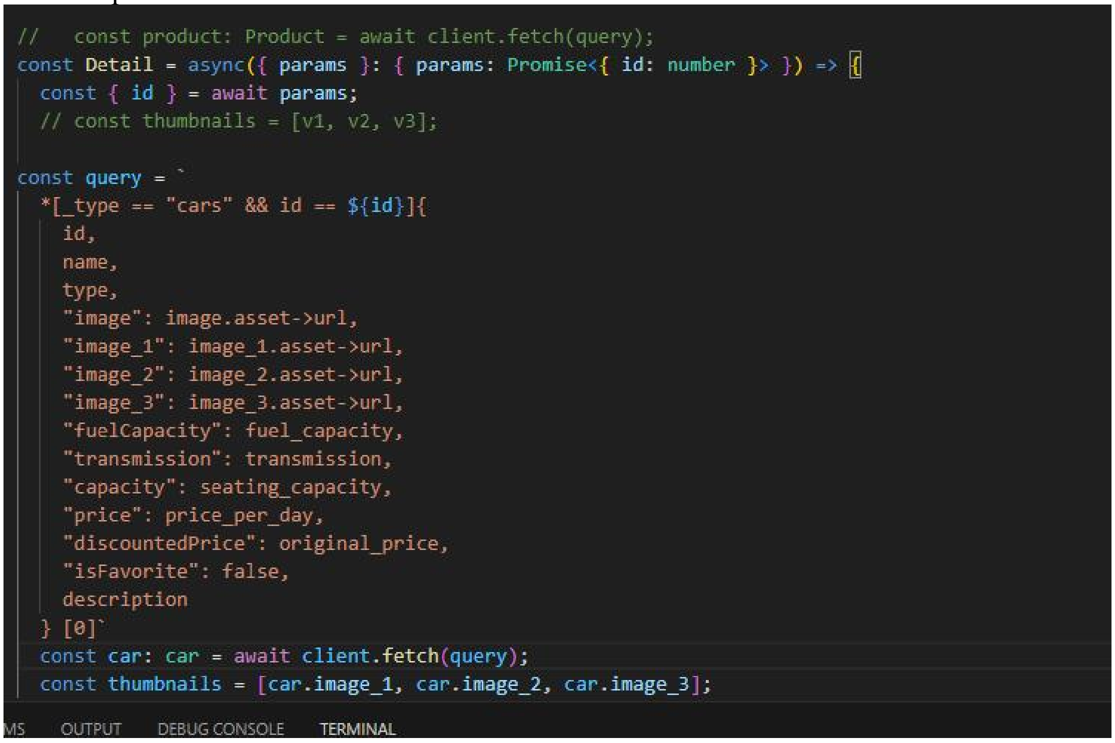
---

4. Back end Integration Refinement

Dynamic Data Handling:

- Back end now uses real-time data for car availability, prices, and booking slots, pulled dynamically from Sanity CMS.

  The integration ensures the from tend reflects up-to-date information without needing manual updates



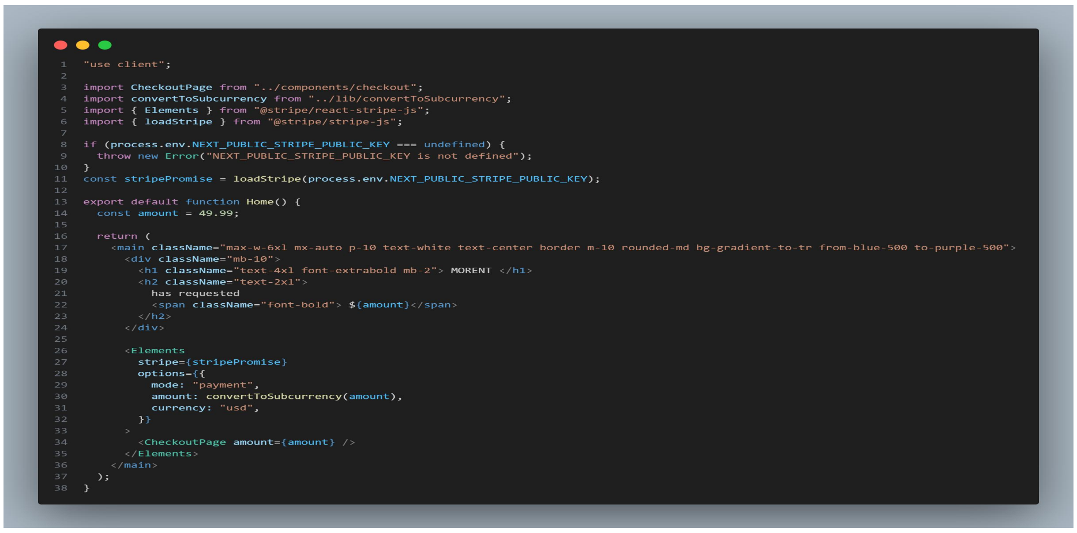
```
//   const product: Product = await client.fetch(query);
const Detail = async({ params }: { params: Promise<{ id: number }> }) => {
  const { id } = await params;
  // const thumbnails = [v1, v2, v3];

const 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];
```

Stripe Payment Integration:

Secure transactions are now processed via Stripe, including payment status verification before confirming bookings.

Error Handling for Payments: Errors like insufficient funds or invalid card details are caught, and users are informed.



```jsx
"use client";

import CheckoutPage from "../components/checkout";
import convertToSubcurrency from "../lib/convertToSubcurrency";
import { Elements } from "@stripe/react-stripe-js";
import { loadStripe } from "@stripe/stripe-js";

if (process.env.NEXT_PUBLIC_STRIPE_PUBLIC_KEY === undefined) {
  throw new Error("NEXT_PUBLIC_STRIPE_PUBLIC_KEY is not defined");
}
const stripePromise = loadStripe(process.env.NEXT_PUBLIC_STRIPE_PUBLIC_KEY);

export default function Home() {
  const amount = 49.99;

  return (
    <main className="max-w-6xl mx-auto p-10 text-white text-center border m-10 rounded-md bg-gradient-to-tr from-blue-500 to-purple-500">
      <div className="mb-10">
        <h1 className="text-4xl font-extrabold mb-2"> MORENT </h1>
        <h2 className="text-2xl">
          has requested
          <span className="font-bold"> ${amount}</span>
        </h2>
      </div>

      <Elements
        stripe={stripePromise}
        options={{
          mode: "payment",
          amount: convertToSubcurrency(amount),
          currency: "usd",
        }}
      >
        <CheckoutPage amount={amount} />
      </Elements>
    </main>
  );
}
```

Booking System Refinement:

Refinement ensures bookings are only confirmed after both payment and car availability are validated.

All booking data (user info, rental details, car model) is stored efficiently in the back end, with associated user profiles for easy management.

---

5. Conclusion

On Day 5, the back end architecture was enhanced with dynamic data integration, error handling, and payment gateway improvements. These changes ensure the platform functions smoothly, providing users with a seamless experience while booking rentals. Testing was key to identifying and addressing potential issues, which were then mitigated for better reliability.