Day 4 - Dynamic Front-end Components - Car Rental

Project Overview

This report documents the implementation of dynamic frontend components for the car rental marketplace. The primary focus was on creating a responsive and dynamic user experience by integrating dynamic data fetching, category filters, search functionality, and other interactive features. Screenshots and code snippets are included to illustrate the results and technical details.

Steps Completed

Step 1: Fetching Dynamic Data

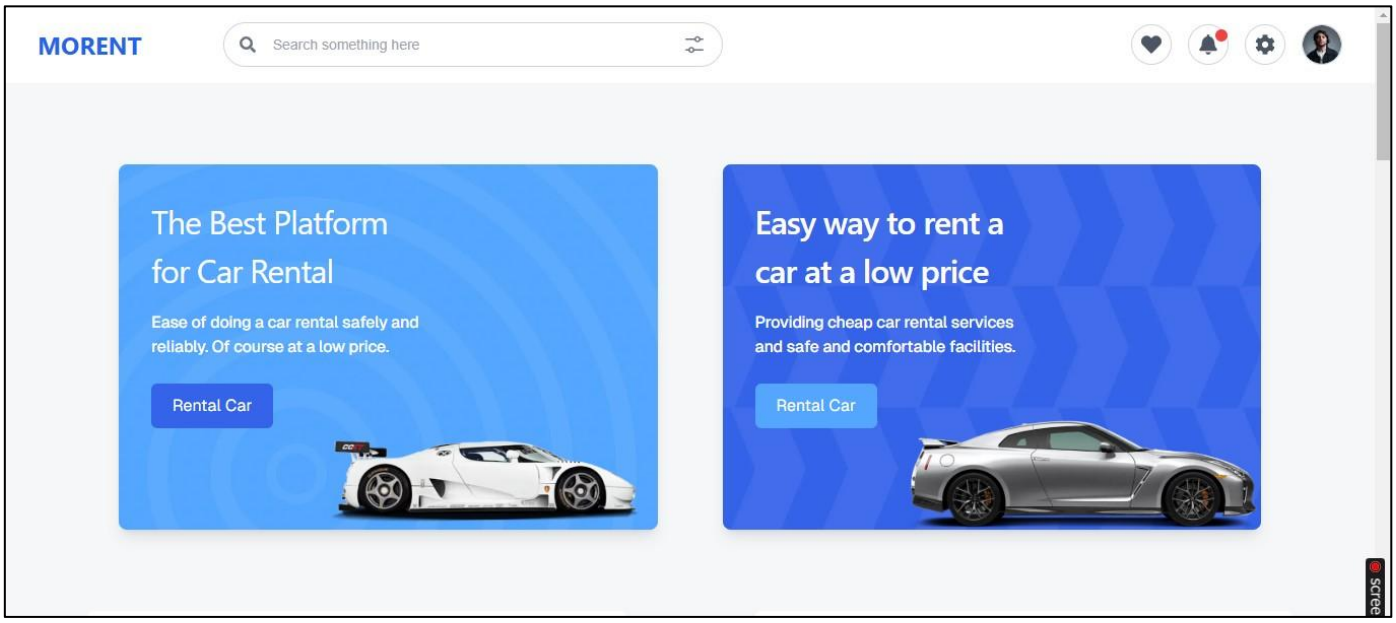
Data Fetching Logic:

a. Implemented a function to fetch car data from an API using fetch.
b. Verified the fetched data by logging responses to the console and testing API endpoints in Postman.

Code Snippet:

```tsx
import Image from "next/image";  16.8k (gzipped: 6.2k)
import Link from "next/link";  32.5k (gzipped: 9.6k)
import { Card, CardContent, CardDescription, CardFooter, CardHeader, CardTitle } from "@/components/ui/card";
import { client } from "@/sanity/lib/client";
import { urlFor } from "@/sanity/lib/image";
import Hero from "@/components/Hero"; // Import Hero component
import PickUpDropOff from "@/components/PickUpDropOff";

interface simplifiedCar {
  id: string;
  name: string;
  type: string;
  image: string;
  fuelCapacity: string;
  transmission: string;
  seatingCapacity: string;
  pricePerDay: string;
  slug: string;
}

Tabnine | Edit | Test | Explain | Document
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;
}

Tabnine | Edit | Test | Explain | Document
export default async function Home() {
  const data: simplifiedCar[] = await getData();
  return (
    <div className="bg-[#f6f7f9] min-h-screen p-4 sm:p-6 lg:p-20 flex flex-col gap-10 font-[family-name:var(--font-geist-sans)]">

      {/* Replace Sections with Hero Component */}
      <Hero />
      <PickUpDropOff />
```

Outcome: Successfully fetched dynamic car data and stored it in the state.



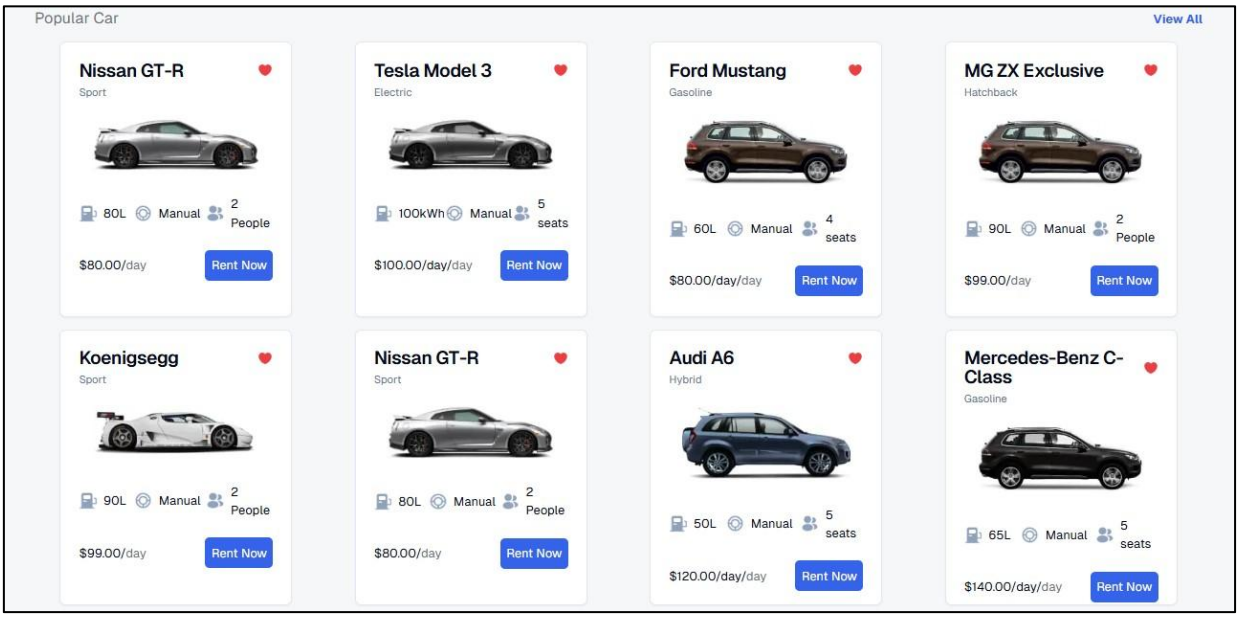
Step 2: Building Dynamic Car List

Component Development: Designed a ProductCard component to display individual car details (e.g., name, price, image).

Code Snippet:

```
export default async function Home() {
  const data: simplifiedCar[] = await getData();
  return (
    <div className="bg-[#f6f7f9] min-h-screen p-4 sm:p-6 lg:p-20 flex flex-col gap-10 font-[family-name:var(--font-geist-sans)]">

      {/* Replace Sections with Hero Component */}
      <Hero />
      <PickUpDropOff />

      <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.map((product) => (
            <div
              key={product.id}
              className="w-full max-w-[304px] mx-auto h-[360px] flex flex-col justify-between hover:scale-105 transition-all duration-300"
            >
              <Card className="h-full flex flex-col">
                <CardHeader>
                  <CardTitle className="w-full flex items-center justify-between">
                    {product.name}{" "}
                    <Image src={"/heart.png"} alt="Heart" 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={product.name}
                    width={220}
                    height={68}
                    className="transition-transform duration-300 transform hover:scale-110"
                  />
                  <div className="flex items-center justify-between gap-4 mt-4">
                    <div className="flex items-center gap-2">
                      <Image
                        src={"/gas-station.png"}
                        alt="Gas Station"
```

Outcome: Successfully fetched dynamic car data list.



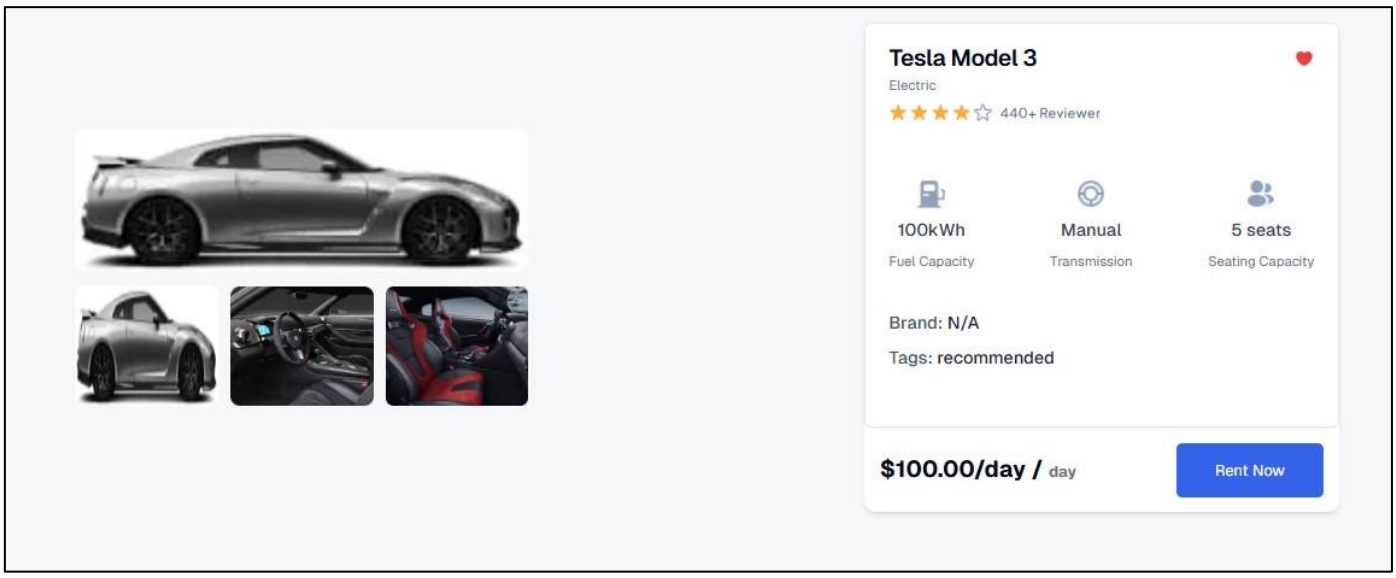
Step 3: Dynamic Car Details Page

Routing Setup: Implemented dynamic routing to navigate to individual car detail pages using /categories/[slug].

Code Snippet:

```
Tabnine | Edit | Test | Explain | Document
async function getData() {
  const query = `*[_type == "car"]{
    _id,
    name,
    type,
    slug,
    image {
      asset -> {url}
    },
    fuelCapacity,
    transmission,
    seatingCapacity,
    pricePerDay
  }`;
  const data = await client.fetch(query);
  return data;
}

interface Car {
  _id: string;
  name: string;
  type: string;
  image: string;
  fuelCapacity: string;
  transmission: string;
  seatingCapacity: string;
  pricePerDay: string;
  brand: string;
  originalPrice: string;
  tags: string[];
  slug: {
    current: string;
  };
}

Tabnine | Edit | Test | Explain | Document
async function getCarBySlug(slug: string) {
  const cleanSlug = slug.replace(/['"]+/g, "");

  const query = `*[_type == "car" && slug.current == "${cleanSlug}"][0]{
    _id,
    name,
    type,
    image,
    fuelCapacity,
    transmission,
```

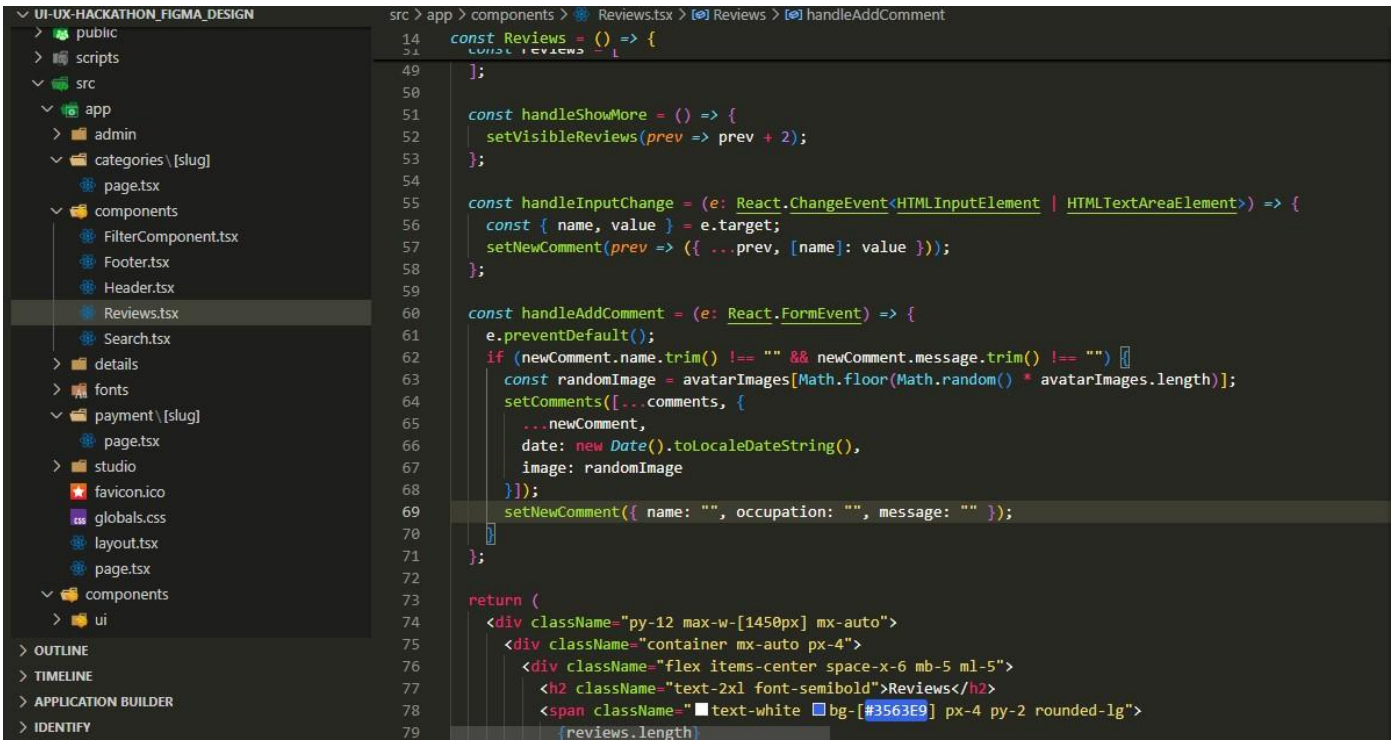
Outcome: Successfully rendered individual car detail pages with accurate routing and dynamic data.



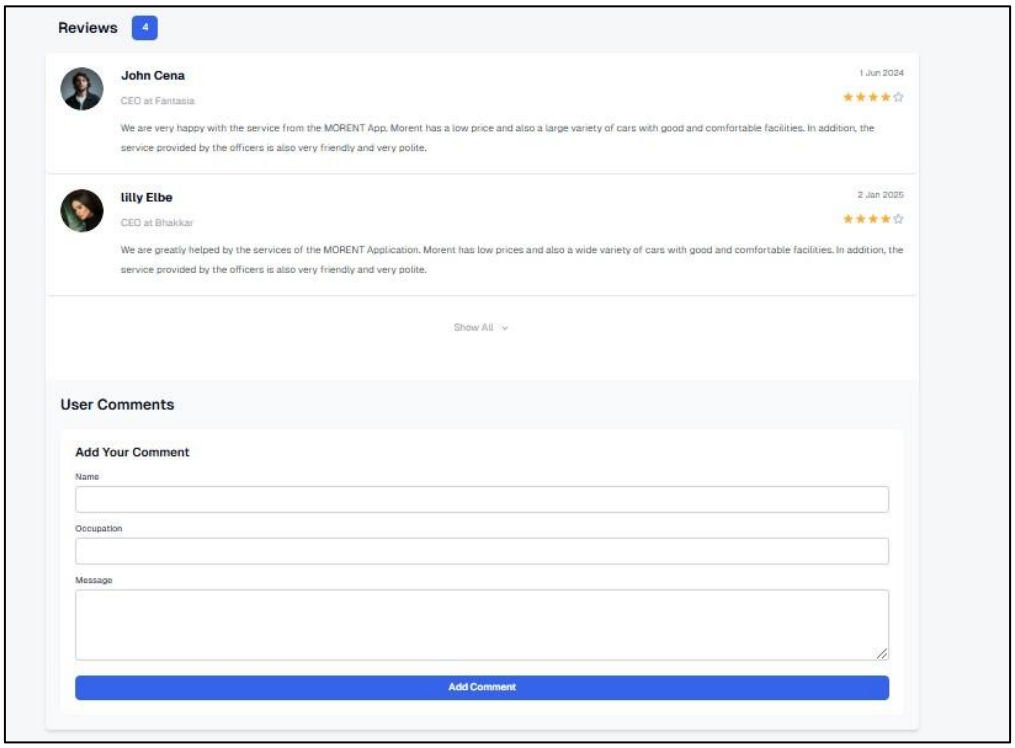
Step 4: Adding Review Component

Review Integration: Created a Reviews component to fetch and display user reviews for each car.

Code Snippet:



Outcome: Displayed user reviews dynamically on the car details page.

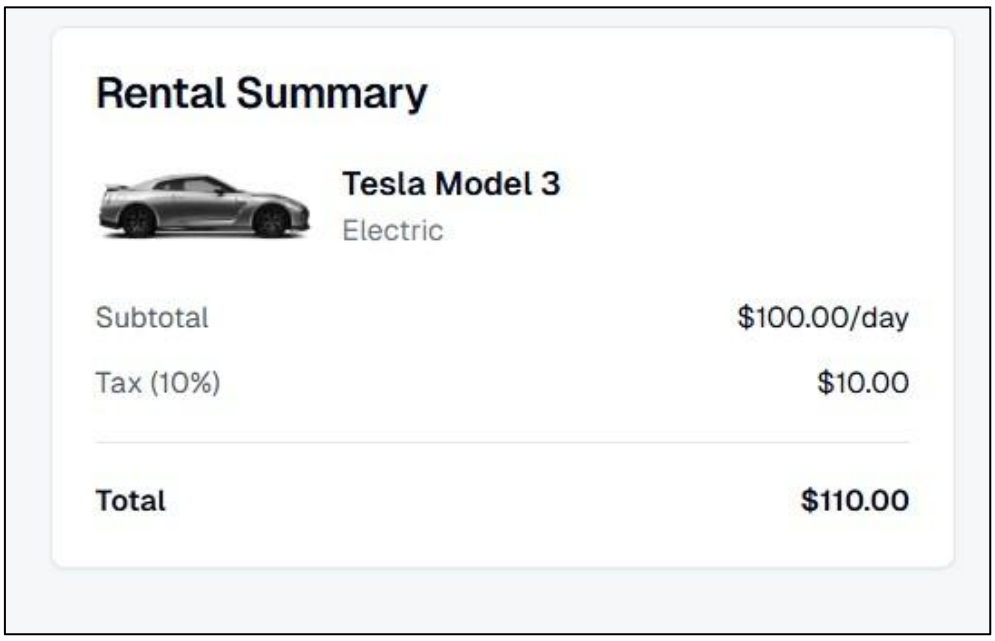


Step 5: Dynamic User History and Rental Summary

User History: Implemented a component to fetch and display a user's rental history dynamically.

Rental Summary: Built a summary component to calculate and display rental details dynamically.

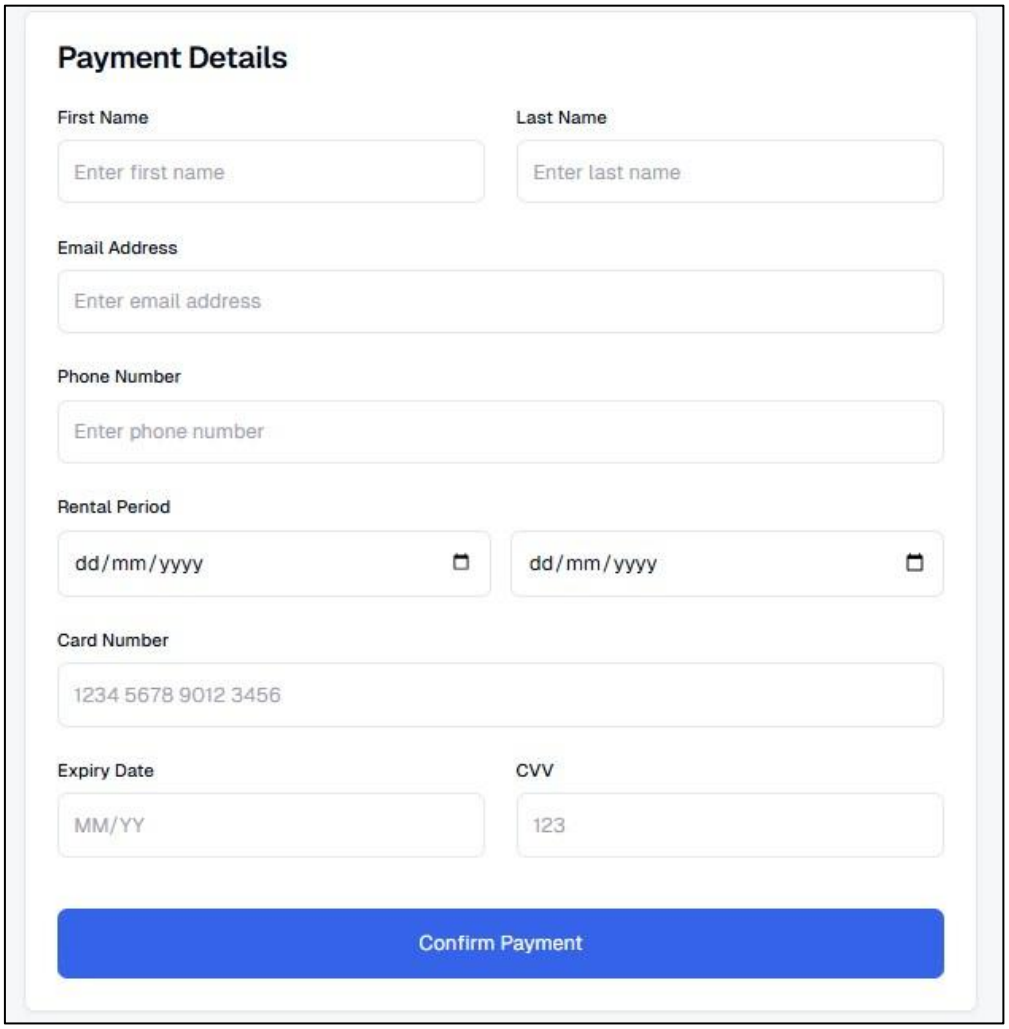
Outcome: Provided users with personalized and dynamic information about their history and ongoing rentals.



Step 6: Dynamic Payment Method

Payment Integration: Added a payment method component to handle user payment options dynamically.

Outcome: Successfully implemented a dynamic payment workflow.



Final Outcome

- **Dynamic Front-end:** Built a responsive, interactive front-end with dynamic car listings, detail pages, and additional features like reviews and user history.
- **Improved User Experience:** Enhanced the user interface with features like pagination, category filters, and search functionality.
- **Technical Success:** Validated all workflows, components, and API integration through thorough testing.

Conclusion

By systematically implementing dynamic components and integrating API data, the project achieved the following milestones:

1. **Dynamic Data Fetching:** Fetched and displayed car data dynamically across the application.
2. **Interactive Components:** Built reusable and dynamic components for car listings, detail pages, and user-specific data.
3. **Enhanced Features:** Added advanced features like reviews, rental summaries, and payment workflows to improve functionality and user satisfaction.
4. **Thorough Validation:** Ensured the accuracy and functionality of all components through testing and debugging.

This work highlights the effectiveness of dynamic front-end development in creating scalable and user-centric applications, setting a strong foundation for further enhancements.

Self-Validation Checklist

✔ Front-end **Components and Development**
Ensure all required front-end components are implemented, functional, and dynamic.

✔ **Styling and Responsiveness**
Verify that the UI is styled attractively and fully responsive across devices.

✔ **Code Quality**
Review the code for readability, consistency, and adherence to best practices.

✔ **Documentation and Submission**
Confirm that all deliverable, including the report, screenshots, and code snippets, are complete and submitted correctly.