Day 4 - Dynamic Frontend Components Report - MORENT

Objective

The focus of Day 4 was to design and develop dynamic frontend components for a marketplace application, fetching data from Sanity CMS or APIs. The emphasis was on modular, reusable, and scalable components with responsive design and an improved user experience.

Key Components Built

1. Car Listing Component

- Rendered cars data dynamically in a grid layout.
- Included the following fields:
    - Cars Name
    - Price
    - Image
    - Fuel Capacity
    - Transmission (e.g: Automatic, Manual)
    - Seat Capacity

Code Screenshot:

```
<Link href={`/cars/${_id}`} className="block group">
  <div className="rounded-lg bg-white p-4 shadow-md h-auto flex flex-col justify-between group-hover:shadow-lg transition-shadow">
    {/* Adjusted Image Section */}
    <div className="my-4 h-[180px] relative">
      <Image
        src={image || "/placeholder.svg"}
        alt={name}
        layout="fill"
        objectFit="contain" // Ensures the full car is visible
        className="rounded-lg"
      />
    </div>
   {/* Price and Rent Now Button */}
   <div className="flex items-center justify-between mt-auto">
     <div>
       <span className="text-lg font-bold text-black-600">PKR {pricePerDay}</span>
       <span className="text-sm text-gray-500">/day</span>
       {originalPrice && (
         <p className="text-sm text-gray-400 line-through">PKR {originalPrice}</p>
       )}
     </div>
     <div className="bg-blue-600 hover:bg-blue-700 text-white px-4 py-2 rounded cursor-pointer">
       Rent Now
     </div>
   </div>
  </div>
</Link>
```

2. Car Detail Component

- Implemented dynamic routing using Next.js for individual car detail pages.
- Included section such as:
    - Reviews
    - Recommended Cars

Code Screenshot::

```
async function getCar(id: string) {
  return client.fetch(`*[_type == "car" && _id == $id][0]`, { id })
}

async function getRecommendedCars() {
  return client.fetch('*[_type == "car" && "recommended" in tags][0...3]')
}

export default async function CarDetailPage({ params }: { params: { id: string } }) {
  const session = await getServerSession()
  const { id } = params // Destructure id from params
  const car = await getCar(id)
  const recommendedCars = await getRecommendedCars()

  return (
    <div className="min-h-screen bg-[#F6F7F9]">
      <Navbar />
      <main className="container mx-auto px-4 py-8">
        <h1 className="text-2xl font-bold mb-6">Car Details</h1>
        <div className="flex flex-col lg:flex-row gap-8">
          <div className="flex-1 space-y-8">
            <Suspense fallback={<Loader />}>
```

3. Search Bar

- Added a search bar to filter products by name.
- Implemented a debounce mechanism for optimized performance.

Code Screenshot:

```
export async function GET(request: Request) {
  const { searchParams } = new URL(request.url)
  const term = searchParams.get("term")

  if (!term) {
    return NextResponse.json({ error: "Search term is required" }, { status: 400 })
  }

  try {
    const cars = await client.fetch(
      `
      *[_type == "car" && name match $term + "*"] {
        _id,
        name,
        "image": image.asset->url
      }
    `,
      { term },
    )

    return NextResponse.json(cars)
  } catch (error) {
    console.error("Error searching cars:", error)
    return NextResponse.json({ error: "An error occurred while searching for cars" }, { status: 500 })
  }
}
```

Steps of Implementation

1. **Setup and Testing**:
    - Confirmed API connectivity and data availability.
    - Used tools to verify API responses.
2. **Component Development**:
    - Modularized components for reusability.
    - Incorporated responsive styling with Tailwind CSS.
3. **State Management**:
    - Utilized React state and context for global and local state management.
4. **Error Handling**:
    - Added fallback UI for scenarios like empty data or failed API requests.

Best Practices Followed

- **Reusable Components**:
    - Ensured components like `CarCard , FilterSidebar, Navbar` and `Footer` could be reused across multiple pages.
- **Responsive Design**:
    - Used tailwind classes for making the marketplace responsive for all devices..
- **Code Quality**:
    - Followed clean coding principles with comments and descriptive variable names.
- **Performance Optimization**:
    - Implemented lazy loading for images and debounce for search functionality.

Challenges and Solutions

1. **Challenge**: API data schema mismatches.
    - **Solution**: Mapped API fields to Sanity schema and added validation during migration.
2. **Challenge**: Handling authentication of users by Google OAuth.
    - **Solution**: Use NextAuth Library for Smooth and error free authentication.
3. **Challenge**: Maintaining responsive design across components.
    - **Solution**: Leveraged Tailwind CSS for quick and efficient styling.

Conclusion

Day 4 focused on building the core frontend components required for a functional and visually appealing marketplace. By applying modern practices and tools, I successfully:

- Created dynamic, reusable components.
- Enhanced user experience with responsive and interactive designs.
- Implemented efficient state and data management strategies.

These skills and implementations align closely with industry standards, preparing me for real-world development challenges.