HACKATHON 03

DAY 02 PLANNING THE TECHNICAL FOUNDATION

OVERVIEW

The purpose of this step is to create a robust and scalable architecture for a car rental e-commerce website. Today's task is to define the technical structure of the project, identify tools, frameworks, and third-party integrations required, and outline the workflows to ensure a smooth development process.

FRONTEND REQUIREMENTS

The frontend will handle the user interface and interaction. It needs to be modern, responsive, and user-friendly.

1. **Framework:**
    - **Next.js:** For its server-side rendering and SEO-friendly capabilities. It will ensure a fast-loading, optimized site.
    - **Tailwind CSS:** To provide responsive and visually attractive styling.
2. **Key Pages:**

Home Page:

- Show a list of available cars for rent with essential details (name, price per day, availability).
- Option to filter or search cars based on user preferences.

II. Car Details Page:

- Displays details of a selected car, including price per day, availability, description, and images.
- Includes a **"Rent Now"** button.

III. Rental Page:

- Allows users to enter details like name, email, rental period, and payment.

IV. Admin Dashboard:

Used for managing car listings, monitoring orders, and customer data.

3. Responsive Design:

- The website must be responsive and work seamlessly on all devices, including desktops, tablets, and smartphones.

BACKEND REQUIREMENTS

The backend will store and manage all data for cars, customers, and rentals.

1. CMS:

Sanity CMS will be used as a backend to handle content like cars, orders, and customer data dynamically.

2. Data Schema:

I. Cars

- `id`: Unique identifier for each car.
- `name`: Name of the car.
- `pricePerDay`: Rental price per day.
- `availability`: Indicates whether the car is available for rent.
- `description`: A brief description of the car.
- `image`: URL for the car image.

II. Orders:

- `orderId`: Unique identifier for each order.
- `productId`: References the car being rented.
- `customerId`: References the customer making the booking.
- `orderDate`: Date of the order.
- `rentalPeriod`: Duration for which the car is rented.
- `totalPrice`: Total cost of the rental.

II. Customers:

- `customerId`: Unique identifier for each customer.
- `name`: Name of the customer.
- `email`: Customer’s email address.
- `phone`: Contact number.
- **`address`**: Customer’s address.

3. API Endpoints:

- `/api/cars`: Fetch car details from Sanity CMS.
- `/api/orders`: Manage rental records.
- `/api/customers`: Retrieve customer details.

THIRD PARTY INTEGRATIONS

To enhance functionality, the project will integrate the following tools:

1. Payment Processing:

- Use **Stripe API** for secure and reliable payment handling.

2. Location Services:

- Integrate **Google Maps** API for selecting pick-up and drop-off locations.

TECHNICAL WORKFLOW

Here's how the website will work:

1. User Browses Cars:

- The user lands on the homepage and views a list of cars dynamically fetched from Sanity CMS.

2. Car Selection:

- The user selects a car to view its details on the car details page.

3. Rental Process:

- The user clicks the **"Rent Now"** button, fills out the rental form, and provides details such as the rental period and contact information. Payment is processed, and the rental is confirmed.

FOLDER STRUCTURE

The project will follow a structured folder system for maintainability:

/project-root

```
/pages
  index.tsx          // Home page
  /cars/[id].tsx     // Dynamic car details page
  /rental.tsx        // Rental page
  /admin/dashboard.tsx // Admin dashboard
/components
  Header.tsx         // Navigation bar
  Footer.tsx         // Footer
  CarCard.tsx        // Component for car cards
/api
  cars.ts            // API for cars data
  orders.ts          // API for orders
/sanity
  schema.ts          // Schema for Sanity CMS
```

Conclusion

This car rental marketplace combines user-friendly design with a robust technical foundation, leveraging Next.js, Sanity CMS, and APIs like Stripe and Google Maps. The scalable architecture and planned workflows ensure a seamless rental experience, making the platform ready for real-world market use.