**TECHNICAL FOUNDATION Hackathon**

**Day 2: Planning the**

**Technical Foundation**

**Recap of Day 1: Business Focus**

**Day 1 Recap: Laying the Marketplace Foundation**

WE are going to fulfill market requirements regarding rent car through online in easy way, where customer find his best car in his desired price, color and model without any hassle, just in few clicks.

**Key Feature**

* Where customer go to our web-application and create an account where his or her details by email then additional detail customer can fill in the input field. And also pay the advance or full payment through payments gateway and our service is to provide car at his door or customer can take car from the said showroom.
* After ended the agreement of rented car we just take car from customer address or customer can drop the car at said show room, during in this business we less the car from our stock after completion the agreement we added the car in our stock dynamically through DATABASE API.

**Primary Purpose Problem Solving Method Without any hassle:**

Designed a platform for urban areas, where people required car for picnic parties or for businesses, offer rent car through online system with convenience, competitive pricing, and reliable delivery with easy payment gateway.

**Target Audience:**

Focused on time-conscious User, families seeking convenience, and professionals looking for reliable delivery options.

**E-Commerce Data Schema:**

1. Core Entities: Products, Customers/User, Orders, Payments, Car Tracker, and Delivery Zones.

2. Key Relationships: Integrated models for real-time tracking, customer order history, and dynamic delivery and stock keeping.

**Marketplace Features:**

1. Dynamic filters for products.

2. Real-time order tracking and flexible payment options.

3. AI-powered personalized recommendations and loyalty programs.

**Day 2 Activities: Transitioning to Technical Planning**

* **Define Technical Requirements:**

This document outlines the technical planning phase for the e-commerce marketplace, focusing on three key areas: frontend requirements, backend integration using Sanity CMS, and third-party API integrations.

1. **Frontend Requirements :**

The frontend will deliver a seamless, user-friendly experience with the following pages and features:

**Essential Pages:**

**Homepage:**

1. Highlights: Featured products, promotional banners, category shortcuts.

2. Call-to-Actions (CTAs): "Rent Now," "Browse Categories," "View Car Details."

**Category Section:**

1. Category Page:

Allow users to browse products by categories (e.g., Sedan, SUV, HatchBack).

1. **Product Listing Page:**

Displays products with:

Filters: Price, category, ratings, availability.

Sorting Options: Best Sellers, Price (Low to High), New Arrivals.

1. **Product Details Page:**

Key Features:

Product name.

Description.

Price.

Stock.

Discount.

Car Type.

Car Capacity.

Car Steering.

Fuel type.

Fuel Tank Storage.

Car Image.

Car Interior Image.

**Car/Cart Page:**

1. Displays selected products with quantity and price by per day.
2. Options to update quantities or remove item(Car).

**Checkout Page:**

1. Captures Customer details with delivery address, payment method, and order summary.

2. Features for applying discount codes and selecting delivery preferences.

**Order Confirmation Page:**

1. Displays order details, estimated delivery time, and shipment tracking.

**About and Contact Pages:**

1. Business details and a contact form for customer inquiries.

**Technical Stack:**

• Frameworks: React.js and Next.js for building dynamic and SEO-friendly pages.

• Component Library: shadcn/ui for customizable, reusable components.

• Styling: Tailwind CSS for responsive and visually appealing design.

1. **Backend with Sanity CMS**

Sanity CMS will serve as the backend to manage dynamic data like products, customers, and orders.

**Sanity Schema Design:**

**Products Schema:**

**Fields:**

* ProductID: Primary Key.
* Name, Description, Category, Price, Stock Quantity.
* Ratings, Reviews, and FAQs.
* Discount (if applicable).

**Customer Schema:**

**Fields:**

* CustomerID: Primary Key.
* Full Name, Email, Phone Number, Address.
* Order History.

**Orders Schema:**

**Fields:**

* OrderID: Primary Key.
* CustomerID: Foreign Key.
* ProductID(s): Many-to-Many relationship.
* Order Date, Status (e.g., Pending, Shipped, Delivered).
* Total Amount.

**Payments Schema:**

**Fields:**

* PaymentID: Primary Key.
* OrderID: Foreign Key.
* Amount Paid, Payment Method (e.g., Credit Card, UPI, Wallet).
* Payment Status (e.g., Successful, Pending).

**Shipment Schema:**

**Fields:**

* ShipmentID: Primary Key.
* OrderID: Foreign Key.
* Courier Service, Tracking Number.
* Estimated Delivery Date, Shipment Status.

**Implementation Steps:**

1. Use Sanity Studio to design and test schemas.
2. Fetch and manipulate data on the frontend using GROQ queries.
3. Optimize schemas for scalability and future expansion.

**3. Third-Party API Integrations**

To provide critical marketplace functionality, integrate the following APIs:

**Payment Gateways:**

**Stripe:**

* Features: Secure payments, support for multiple payment methods, and real-time transaction updates.
* Integration: Use Stripe SDKs and APIs for seamless integration.

**PayPal:**

* Features: Widely accepted payment solution with options for credit/debit card payments and wallets.
* Integration: Use PayPal’s REST API for transactions.

**Additional APIs:**

* **Google Maps API:**

Use Case:Address validation and delivery zone mapping.

* **Notification APIs (Email/SMS):**

Use Case: Send order confirmations and delivery status updates

**Middleware Implementation:**

• Use Node.js and Express.js to handle API requests and process server-side logic.

• Secure API endpoints using JWT (JSON Web Tokens).

**Development Pipeline**

1. Frontend Development:

Build responsive pages using React.js, Next.js, and Tailwind CSS.

1. Backend Development:

Implement schemas in Sanity CMS and connect the frontend via APIs.

1. API Integration:

Integrate payment and shipment APIs to ensure seamless functionality.

1. Testing:

Conduct thorough testing for functionality, responsiveness, and security.

1. Deployment:

Host the platform on Vercel (frontend) and Heroku or AWS Lambda (backend).

This detailed plan aligns technical implementation with business goals, ensuring a scalable, user-centric, and efficient e-commerce platform.

1. **Plan API Requirements**

Here is a table summarizing the API endpoints for the E-Commerce platform:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Endpoint Name** | **Method** | **Description** | **Request Body** | **Response Example** |
| /api/users/register | POST | Register a new user | { "username": "john", "email": "john@example.co m", "password": "pass123" } | {"status": "success", "message": "User registered successfully.} |
| /api/users/login | POST | Authenticates a user and generates a JWT. | { "email": "john@example.co m", "password": "pass123" } | { "status": "success", "token": "jwt.token.here" } |
| /api/users/{id} | PUT | Updates user details. | {"username": "john\_updated", "email": "updated@example .com" } | {"status": "success", "message": "Profile updated successfully." } |
| /api/categories | GET | Retrieves a list of all product categories. | None | { "status": "success", "data": [ { "id": 1, "name": "Groceries" } ] } |
| /api/categories/{id}/products | GET | Fetches products within a specific category. | None | { "status":"success", "data": [ { "id": 101, "name": "Apples" } ] } |
| /api/products/{id} | GET | Retrieves details of a specific product. | None | { "status": "success", "data": { "id": 101, "name": "Apples" } } |
| /api/cart/add | POST | Adds a product to the user's cart. | {"product\_id": 101, "quantity": 2 } | {"status": "success", "message": "Item added to cart." } |
| /api/cart | GET | Retrieves the current state of the user's cart. | None | { "status": "success", "data": { "items": [ { "id": 101 } ] } } |
| /api/checkout | POST | Processes payment and places an order. | {"payment\_metho d": "card", "shipping\_addres s": { ... } } | {"status": "success", "message": "Order placed successfully." } |
| /api/orders/{id} | GET | Retrieves the details of a specific order. | None | { "status": "success", "data": { "order\_id": 12345 } } |
| /api/homepage | GET | Retrieves homepage content, including featured items. | None | { "status": "success", "data": { "featured\_produc ts": [ ... ] } } |
|  |  |  |  |  |

This table provides an at-a-glance overview of the API endpoints, their purposes, and example data formats for requests and responses.

**4. Write Technical Documentation**

**Technical Documentation for eCommerce System:**

This documentation provides a comprehensive guide to the eCommerce system's architecture, workflows, API endpoints, and Sanity CMS schema examples. It aims to assist developers, project managers, and stakeholders in understanding and implementing the platform effectively.

1. **System Architecture Overview**

The eCommerce platform is built with a modular architecture to ensure scalability, maintainability, and performance.

**Frontend:**

* + - Framework: React.js / Next.js for fast and responsive UI.
    - Styling: TailwindCSS or Material-UI for component-based styling.
    - State Management: Redux or Context API for seamless state handling.

**Backend**:

* + - Framework: Node.js with Express.js for API creation.
    - Database: MongoDB for a scalable, NoSQL solution to store products, users, and orders.
    - Authentication: JSON Web Tokens (JWT) for secure user authentication and authorization.

**CMS:**

* + - Sanity.io: For managing dynamic content such as categories, product descriptions, and blogs.

**Deployment:**

* + - **Hosting:** AWS, Vercel, or Netlify for deployment.
    - **CI/CD:** GitHub Actions or Jenkins for automated deployment pipelines.

**2. Key Workflows**

**2.1 User Registration & Authentication**

1. **User Signup:** Users register by providing their email, password, and profile details. Data is stored in the database after validation.
2. **Login:** Users enter their credentials to obtain a JWT token, enabling secure session handling.
3. **Password Recovery:** Users reset passwords via a token-based recovery system.

**2.2 Product Browsing & Filtering**

1. Users view categories fetched from the CMS.

2. Clicking a category triggers the /categories/{id}/products API to display relevant products.

3. Users can filter products by attributes such as price, ratings, or availability.

**2.3 Cart Management**

1. Users add products to their cart via the /cart/add endpoint.

2. The cart updates dynamically, storing items in the database or local storage for guest users.

3. The cart is displayed using the /cart endpoint.

**2.4 Checkout & Payment**

1. The user proceeds to checkout, providing payment and shipping details.

2. The /checkout endpoint processes the payment and creates an order.

3. Users receive order confirmation via email.