Name : Areeba Awan

Roll : 00263538

Rising Star Q 2

Hackathon 3 Day 2 Task

MARKETPLACE TECHNICAL FOUNDATION

DAY - 2

GOAL : TRANSITION FROM BUISNESS PLANNING TO TECHNICAL IMPLEMENTAION

Comforty Our NonStop E-Commerce Store

Objectives:

This Document contains all details about my E-commerce Marketplace focusing on key steps .It is tailored for building a robust and user-friendly platform where people can get a high-quality and luxurious furniture with a competitive prices for their homes and offices. This Technical Planning follows the brainstorming from Hackathon Day 1.

My Business Goal:

I'm planning to build a Fully Responsive and Eco-Friendly Website for Comfortable Furniture like Sofas, Chairs, Stools, Benches, Desks, Arm Chairs Etc. Here's the details that it will include:

Step 1 : Front-End Key Requirements:

1. Set up The Environment for Front-End

- NextJs : Frame-work for building the Application
- Tailwind Css : For Responsive Design And Styling.
- Typescript : For Functionality and Interactions.
- React Icons : Used For Icons based on the website.
- Redux : For Managing Global State Easily (Cart Data, User Data).
- Figma : For UI/UX Designs
- Hosting and Deployment: **Vercel (for frontend)**

User-Friendly Interface: Comforty will use a NextJs for Front-End {User Interface} to give a best layout that makes customers to browse and find the perfect products.

Responsive Design:

Comforty will Feature a mobile design as a User's first approach , then user on Desktop and other devices will also use my platform.It will ensuring that the site looks great and working properly with the flexible grids and responsive images on all devices.

Essential Pages:

Sign-up page : Creating a **sign-up/Login page** involves a form where users can input their information to create an account.

Start with Home Page: Then User will Go to Home Page (A Welcoming Page That Shows Featuring Products)

About Page: **About Us** content emphasizes Comforty's values, mission, and offerings. It's written to engage visitors, giving them a sense of the company's purpose while highlighting its key products and commitment to quality and sustainability.

Shop Page: A Shop Page is created to showcase all the featured Products.

Product Listing Page: Display Products with Clean and organize way that user can easily browse.

Product Details Page : Product Details Page Offers in-depth information about each item, including comfort features, materials, dimenstions, sizes and high quality images.

Cart Page: Ensures that users will add, modify and remove products before proceeding to checkout.

Checkout Page: A simple and secure process to review the order, select payment, and finalize the purchase.

Order Confirmation Page: It reassures customers that their order has been successfully placed and gives them all the important details about their purchase.

WishList Page: A **Wishlist** page is a great feature for e-commerce websites. It allows users to save items they're interested in, making it easier to purchase them later. Here's a detailed layout and sample content for a **Wishlist** page, tailored to a brand like **Comforty**.

Faqs Page : A well-structured **FAQs** (Frequently Asked Questions) page is essential for any e-commerce website, as it helps customers find answers to their common queries quickly.

Back-end Key Requirements:

- Content Management System (Sanity)
- Order Tracking and Shipment: (ShipEngine)
- Database: MongoDB (for authentication)

- AWS (for backend)
- Payment Gateway: Stripe, paypal

Overview of backend:

- REST APIs to manage users, products, orders, and delivery zones.
- Handles business logic, data validation, and integration with external services

DATABASE (MONGODB)**:**

- **NoSQL Database**: MongoDB is used to manage flexible and scalable data structures, ideal for Comforty's evolving product catalog, customer profiles, and order histories.

CONTENT MANAGEMENT SYSTEM (CMS) - SANITY**:**

- **Dynamic Content**: Sanity is utilized to easily manage dynamic content such as banners, promotions, featured products, blog posts, customer testimonials, and more on the Comforty website.

ORDER TRACKING (SHIPENGINE):

- **Real-time Order Tracking**: Integrated with ShipEngine for real-time shipment tracking. Customers can view the status of their orders (shipped, in-transit, delivered) directly from their account or through automated emails.

AUTHENTICATION (MONGODB):

- **User Credentials**: MongoDB securely stores user credentials for account creation and login.
- **Secure Authentication**: Passwords are stored securely using bcrypt for hashing to prevent sensitive data exposure.

Deployment:

The **backend** is deployed on **AWS Lambda** using a **serverless architecture**. This approach allows Comforty to only pay for what is used, minimizing costs and scaling the service efficiently.

THIRD-PARTY API:

1. SHIPMENT TRACKING API INTEGRATION: SHIPENGINE API

ShipEngine is an excellent API choice for shipment tracking as it allows users to track the status of their orders in real-time. We'll integrate ShipEngine to manage and track the status of each order.

Steps for Integration:

- **Get an API Key**: You need to sign up for ShipEngine and get an API key.
- **Track Orders**: Once orders are shipped, you can fetch the status by calling the tracking API endpoint.

Frontend Integration:

- Display real-time tracking status on the user's order page or via email notifications.
- Customers can view order tracking information such as "In Transit," "Delivered," or "Out for Delivery."

Backend Integration:

- **API Endpoint** to fetch tracking data using the shipment number.
- **Store Tracking Information** in MongoDB for each order, so customers can see the full history of their shipments.

2. Payment Processing (Stripe, Jazz Cash, EasyPaisa, Kuickpay):

- **Integration:** Secure payment processing with multiple gateways.
- **API Endpoint:** Payment-related endpoints for handling transactions, including Cash on Delivery (COD) option.
- **Outcome:** Orders processed only after successful payment confirmation or COD selection.

3. Order Management:

- **Database:** MongoDB stores order data (customer ID, product ID, quantity, status).
- **API Endpoint:** POST /orders to create orders (status defaults to "Pending").
- **Outcome:** Order information processed and stored for tracking. Note: Orders cannot be edited once created.

API Endpoints

User Management

- POST /api/auth/register: Register a new user.
- POST /api/auth/login: User login.
- GET /api/users/profile: Fetch user profile (requires authentication).
- PUT /api/users/update: Update user details.

Product Management

• GET /api/products: List all products.

• GET /api/products/:id: Fetch product details by ID.

• POST /api/products: Add a new product (requires seller role).

• PUT /api/products/:id: Update product details (requires seller role).

• DELETE /api/products/:id: Delete a product (requires seller role).

Order Management

• POST /api/orders: Create a new order.

• GET /api/orders: List all orders for the authenticated user.

• GET /api/orders/:id: Fetch details of a specific order.

Category Management

• GET /api/categories: List all categories.

• POST /api/categories: Add a new category (requires admin role).

• PUT /api/categories/:id: Update category details (requires admin role).

• DELETE /api/categories/:id: Delete a category (requires admin role).

Payment Management

• POST /api/payments: Initiate a payment.

• GET /api/payments/status: Fetch payment status. Shipment Management

• POST /api/shipments: Create a new shipment. • GET /api/shipments/track: Track shipment status.

Data Schema Updates

Users:

- user_id: Unique identifier for the user.
- username: User's full name.
- email: User's email address.
- password_hash: Encrypted password.
- role: Role of the user (admin, seller, customer).
- order_ids: List of IDs referencing the user's orders.
- product_ids: List of IDs referencing products added by the user (if seller).

Products:

- product_id: Unique identifier for the product.
- name: Name of the product.
- stock: Availability status of the product.
- description: Detailed description of the product.
- image_url: URL of the product image.
- sizes (optional): Available sizes for the product.
- user_id (mandatory): ID of the seller who listed the product.

Orders:

- order_id: Unique identifier for the order.
- customer_id: Reference to the customer placing the order.
- product_id: Reference to the rented product.
- quantity: Number of products rented.
- status: Current status (e.g., Pending, Confirmed, Completed).
- order_date: Timestamp of when the order was placed.

Sellers:

- seller_id: Unique identifier for the seller
- name: Full name of the seller.
- email: Email address of the seller.
- products: List of product IDs listed by the seller.
- delivery_zones: List of delivery zones managed by the sellers.

Delivery Zones:

- zone_id: Unique identifier for the delivery zone.
- zone_name: Name of the delivery area.
- coverage_area: Geographic coverage of the delivery zone

- drivers: List of drivers assigned to the zone.

Relationships between them:

1.User and orders:

- One user can have multiple orders (One-to-Many relationship).

2. User and Products:

- One user can list multiple products (One-to-Many relationship).

3.Orders and Products:

- One order can include multiple products, and each product can be part of multiple orders (Many-to-Many relationship).

4. *Seller and Products:*

- One seller can list multiple products (One-to-Many relationship).

5. *Seller and Delivery Zones:*

- One seller can manage multiple delivery zones, and one delivery zone can have multiple sellers (Many-to-Many relationship).

6. *Payments and Orders:*

- Each payment is associated with exactly one order (One-to-One relationship).

7. Delivery Zones and Drivers:

- One delivery zone can include multiple drivers (One-to-Many relationship).

WorkFlows

Clerk (Authentication)

• Purpose: Clerk handles user authentication and authorization for your marketplace. It provides features like sign-up, login,and user profile management, ensuring secure access to your platform.

• Why Use It: Simplifies the implementation of secure authentication and ensures smooth user registration and login
experiences.

Sanity CMS (Content Management)

• Purpose: Sanity CMS serves as the backend for managing content such as product data, customer details, and orders. It allows for flexible and scalable management of product listings and other dynamic content.

• Why Use It: Provides a powerful content management system that integrates seamlessly with your frontend, allowing easy updates and real-time data management.

ShipEngine API (Shipping and Tracking)

• Purpose: ShipEngine API provides shipping label creation, real-time tracking, and rate calculation across multiple carriers(e.g., FedEx, UPS, USPS).

• Why Use It: Simplifies logistics by allowing you to easily integrate shipment tracking and management, offering your customers real-time updates on their orders.

Stripe (Payment Gateway)

• Purpose: Stripe is a secure payment processing platform that enables you to accept payments from customers through various methods (credit cards, PayPal, etc.).

• Why Use It: Offers a seamless and secure checkout experience, with robust support for different currencies and payment methods, while also handling fraud prevention and chargebacks.

React Context API (Cart Functionality)

• Purpose: The React Context API is used to manage the state of the shopping cart, including adding/removing items, displaying the total, and maintaining cart details across pages.

• Why Use It: Allows you to efficiently manage and share cart data across different components in your React app, creating asmoother user experience with real-time updates.

Tools & Libraries:

✓ Clerk: Authentication.
✓ Sanity CMS: Content management.
✓ ShipEngine API: Shipping and tracking.

✓ Stripe: Payment gateway.
✓ React Context API: Cart functionality.

Security Considerations

Data Encryption:

- Use HTTPS for all communications.
- Encrypt sensitive user data (e.g., passwords).

2. Authentication and Authorization:

- MongoDB stores and validates credentials securely.
- Role-based access control for admin and users.

3. Payment Security:

- Use PCI-compliant Stripe APIs for payment processing

4. API Security: a. Rate limiting to prevent abuse.

- Input validation to avoid SQL injection and XSS.