Name:Azka kiran

Slot: Thursday Morning (9am to 12pm)

Roll no: 00310534

Hackathon 3 - Day 2

Planning The Technical Foundation

Marketplace Type : E-Commerce

Installation:

- Node Js
- Vs code
- Next Js
- Typescript
- Tailwind CSS
- Sanity.io
- Shadcn.ui
- React Icons
- Stripe
- Animation library

User Interface:

- Home Page
- About page
- Shop Page
- Product Listing Page
- Product Detail Page
- Cart Page
- WishList Page
- Contact Page
- FAQs Page

Functionalities:

- Browser Product
- Add To Cart
- Add To Wishlist
- Checkout
- Shipping Tracker
- Inventory
- Real Time Updates Inventory

Executive Summary:

- **Objective**: Develop A Robust E-Commerce Platform To Empower Small Businesses And Individual Sellers In The Furniture Industry.

  Sales Data: Comprehensive insights into the number of items sold, revenue generated, and best-selling products.

  Customer Trends: Analysis of customer preferences, purchase behavior, and feedback.

  Inventory Management: Real-time updates on stock levels, helping sellers manage inventory efficiently.

  Scope:

    Platform Architecture: Design and development of Scalable, user-friendly e-commerce system.

    Core Features: Integration of essential functionalities such as product listings, search and filter options, secure payment processing, and user accounts.

    Seller Tools: Dedicated features for small businesses and individual sellers, including inventory management, sales analytics, and order tracking.

    Customer Experience: Enhancing user experience with intuitive navigation, personalized recommendations, and responsive customer support.

- This document outlines the technical approach, key features, and integration of ideas from Hackathon Day 1 and Day 2 recommendations.

Introduction:

- **Purpose**: To present the technical strategy for building a furniture-focused e-commerce marketplace.

Target Audience:

Homeowners and Renters: Individuals looking to furnish

Decorate their living spaces with quality furniture.

Interior Designers: Professionals seeking unique and diverse furniture pieces for their design projects.

Small Businesses: Furniture retailers and boutique shops aiming to reach a broader online audience.

Furniture Enthusiasts: Consumers interested in custom, vintage, or sustainable furniture options.

Commercial Buyers: Businesses, such as offices, restaurants, and hotels, needing bulk furniture purchases.

Technical Architecture:

1. **Backend**:
   - Use of Node.js/Express for scalable server-side applications.
   - Integration with a robust database (e.g., PostgreSQL, MongoDB) for managing product listings, user accounts, and transactions.
2. **Frontend**:
   - React/Typescript for a dynamic, responsive user interface.

- Integration with design frameworks like Material-UI or Tailwind CSS for a modern look.

3. **Database (MongoDB):**

    a. SQL database to manage flexible and scalable data structures.

    b. Collections for products, orders, customers, delivery zones, and user authentication.

4. **Sanity (CMS):**

    a. Manages dynamic content like (banners, featured products, and blog posts)

5. **Order Tracking (ShipEngine):**

    a. Tracks orders in real time.

    b. Manages shipment and delivery updates.

6. **Authentication (MongoDB):**

    a. MongoDB stores user credentials securely.

    b. Passwords encrypted with hashing algorithms.

7. **Deployment:**

    a. Frontend deployed on Vercel.

    b. Backend deployed on AWS Lambda.

8. API Gateway:

    a. Handles incoming API requests and routes them to the

    Appropriate backend service

b. Provides a single entry point for API requests

9. Third-Party Services:

a. Stripe for payment processing

b. Twilio for SMS and notification services

c. Sanity CMS for content management

Core Features:

1. Seller and Buyer Accounts

- **Seller Dashboard**:
    - Features for product listing, inventory management, and sales tracking.
- **Buyer Dashboard**:
    - Wishlist, order tracking, and personalized recommendations.

Product Management:

- **Cataloging**:
    - Advanced categorization for various furniture types.
    - Search and filter options based on (material, price, size, colors, description) etc.

Payment and Security:

- **Payment Gateway Integration**:
    - Support for multiple payment options like credit cards, digital wallets, and bank transfers.

- **Security**:
    - Implementation of SSL, data encryption, and compliance with GDPR for user data protection.

System Workflow Work:

1. **User Signup/Login:**

    a. Input: User credentials (email, password).

    b. Database: MongoDB for storing user data securely with hashed passwords.

    c. API Endpoint: POST /register, POST /login, and GET /verify-route for handling user authentication and verification

    d. Outcome: JWT token issued for session management.

2. **Content Management (Sanity CMS):**

    a. Admin Role: Manages product listings, banners, and blog content.

    b. API Integration: GROQ Qeries to fetch content dynamically for frontend.

    c. Outcome: Content stored and updated in Sanity is rendered seamlessly on the Next.js frontend.

3. **Product Browsing and Checkout:**

    a. Frontend: Next.js provides server-side rendering for product pages.

    b. Database: MongoDB stores product details (name, price, stock, description, sizes, etc.).

c. API Endpoint: GET /products for listing, GET /products/:id for details, and POST /products to add products (admin/seller role only).

d. Outcome: Users browse, add products to cart, and proceed to checkout.

4. Order Management:

a. Database: MongoDB stores order data (customer ID, product ID, quantity, status).

b. API Endpoint: POST /orders to create orders (status defaults to "Pending").

c. Outcome: Order information processed and stored for tracking. Note: Orders cannot be edited once created.

5. Shipment Tracking (ShipEngine):

a. Integration: ShipEngine API for real-time shipment tracking.

b. API Endpoint: GET /shipments/: order ID to fetch delivery status.

c. Outcome: Users receive real-time updates on their order delivery.

6. Payment Processing (Stripe, Jazz Cash, Easy Paisa Bank):

a. Integration: Secure payment processing with multiple gateways.

b. API Endpoint: Payment-related endpoints for handling transactions, including Cash on Delivery (COD) option.

c. Outcome: Orders processed only after successful payment confirmation or COD selection.

API Endpoints:

Product Management:

• GET /Api/products: List all products.

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

• POST /Api/products: Add a new product (require seller role).

- PUT /Api/products/id: Update product details (requires seller role).
- DELETE /Api/products/id: Delete a product (requires seller role).

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.

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.

Sanity schema:

```
export const product = {
                    productId: "product",
                    name: "document",
                     title: "Product",
                    descripton: " document detail",
                    price: "number",
                    image:(url),
                    stock: " number",
                    fields: [
                {
                    name: "image",
                    title: "Product Image",
                    descripton: " document detail",
        price: "number",
        image:(url),
        stock: " number",


},
{
        name: "name",
         title: "Product Title",
        descripton: " document detail",
        price: "number",
        image:(url),
        stock: " number",
},
{
         name: "price",
        title: "Product Price",
        descripton: " document detail",
        price: "number",
        image:(url),
        stock: " number",
},
{
        name: "price_id",
          title: "Stripe Price ID",
         descripton: " document detail",
         price: "number",
         image:(url),
         stock: " number",
},
{
         name: "description",
          title: "Product Description",
         descripton: " document detail",
         price: "number",
         image:(url),
         stock: " number",
},
{
         name: "slug",
         type: "slug",
         title: "Product Slug",
         descripton: " document detail",
         price: "number",
         image:(url),
         stock: " number",
          options: {
         source: "name",
  },
         },
{
          name: "stock",
          type: "number",
         title: "Stock",
         descripton: " document detail",
         price: "number",
         image :(url),
         stock: " number",

},
],
};
```

Integration Details:

Sanity CMS:

• **Used to manage dynamic content such as:**

Homepage banners.

Category highlights.

Blog posts for marketing.

Sanity's GROQ Query API will be used to fetch content dynamically.

ShipEngine:

• **API used to**: Generate shipping labels.

Track shipments.

Provide real-time delivery update

Relationships:

1. User and Orders:

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

2. User and Products:

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

3. Orders and Products:

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

4. Seller and Products:

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

5. Seller and Delivery Zones:

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

6. Payments and Orders:

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

7. Delivery Zones and Drivers:

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

Relationships Between Models User - Products:

One-to-Many: A user (seller) can list multiple products.

User - Orders: One-to-Many: A customer can place multiple orders. **Product - Orders:** Many-to-Many: An order can have multiple products, and a product can be part of multiple orders.

Category - Products: One-to-Many: A category can have multiple products.

Order - Payments: One-to-One: An order can have only one payment.

Security Method:

1. Data Encryption:

a. Use HTTPS for all communications.

b. Encrypt sensitive user data (e.g., passwords).

2. **Authentication and Authorization:**

a. MongoDB stores and validates credentials securely.

b. Role-based access control for admin and users.

3. Payment Security:

a. Use PCI-compliant Stripe APIs for payment processing.

4. API Security:

a. Rate limiting to prevent abuse.

b. Input validation to avoid SQL injection and XSS.

Conclusion:

This technical plan provides a strong foundation for the furniture marketplace, using modern technologies to create a smooth and scalable platform for both small businesses and customers. It focuses on delivering a great user experience and supporting business growth, making shopping easy and enjoyable for everyone.