Technical Planning Documentation Overview

This document outlines the technical plan for developing an E-Commerce Marketplace to empower small businesses and individuals by providing a platform to sell their products online. The technical planning follows the brainstorming from Hackathon Day 1 and incorporates the recommendations from the Day 2 guidelines.

Key Technologies

- **Frontend:** Next.js
- **Content Management System (CMS):** Sanity
- **Order Tracking and Shipment:** ShipEngine
- **Hosting and Deployment:** Vercel (for frontend)

Technical Architecture

System Overview

1. **Frontend (Next.js):**
   a. Client-side rendering for speed and responsiveness.
   b. Server-side rendering for SEO and product page preloading.
   c. Integration with Sanity CMS for dynamic content.
2. **CMS (Sanity):**
   a. Manages dynamic content like banners, featured products, and blog posts.
3. **Order Tracking (ShipEngine):**
   a. Tracks orders in real time.
   b. Manages shipment and delivery updates.
4. **Authentication (clerk):**
   a. Clerk stores user credentials securely.
5. **Deployment:**
   a. Frontend deployed on Vercel.

System Components and Workflow

1. **Content Management (Sanity CMS):**
   a. Admin Role: Manages product listings, banners, and blog content.
   b. API Integration: GROQ Queries to fetch content dynamically for frontend.

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

2. **Product Browsing and Checkout:**
   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.
3. **Shipment Tracking (ShipEngine):**
   a. Integration: ShipEngine API for real-time shipment tracking.
   b. API Endpoint: GET /shipments/:orderId to fetch delivery status.
   c. Outcome: Users receive real-time updates on their order delivery.
4. **Payment Processing (Stripe, Jazz Cash, EasyPaisa, Kulckpay):**
   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.

User Management

a. I will use a clerk for my user management system.

Product Management

- GET /api/products: List all products.
- GET /api/products/:id: Fetch product details by ID.
- 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.

Component Details and Interactions

- **Frontend (Next.js):**
  o Handles user interactions and renders data fetched via APIs.
  o Communicates with the backend for authentication, product data, and order processing.
- **Sanity CMS:**
  o Manages dynamic content, ensuring marketing and product information stays up-to-date.

Data Schema Updates

Products:

- product_id: Unique identifier for the product.
- name: Name of the product.
- price: Rental cost per day/hour.
- 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.

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.

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 seller.

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).

Integration Details
Sanity CMS

- Used to manage dynamic content such as:
  o Homepage banners.
  o Category highlights.
  o Blog posts for marketing.

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

ShipEngine

- API used to:
  o Generate shipping labels.
  o Track shipments.
  o Provide real-time delivery updates.

Stripe Integration

- Used for:
  o Processing payments securely.
  o Managing subscriptions (if applicable).
  o Handling refunds and payment disputes.

Deployment Plan
Frontend (Next.js)

- **Hosting:** Vercel.
- **CI/CD:** Automatically deploy changes from the GitHub repository.

Database (sanity)

- **Hosting:(vercel)**.

Security Considerations

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.

Monitoring and Maintenance

1. **Monitoring Tools:**
   a. New Relic for application performance.
   b. CloudWatch for serverless logs.
2. **Error Tracking:**
   a. Sentry for real-time error tracking and debugging.
3. **Maintenance:**
   a. Weekly database maintenance and optimization.
   b. Regular updates for dependencies to fix vulnerabilities.

Conclusion
This technical plan ensures a robust foundation for the marketplace, leveraging modern technologies to deliver a seamless and scalable platform for small businesses and customers alike