Day 2

Technical Planning Documentation Overview

Overview

This document details the technical strategy for developing an ECommerce Food Marketplace aimed at empowering local food businesses and independent chefs by providing a platform to sell their food products online. The planning follows the brainstorming sessions from Hackathon Day 1 and integrates key recommendations from Day 2.

Key Technologies

- Frontend: Next.js
- Content Management System (CMS): Sanity
- Order Tracking and Delivery: ShipEngine
- Database: MongoDB (for authentication)
- Hosting and Deployment: Vercel (for frontend
- Payment Gateway: Stripe

Technical Architecture

System Overview

1. Frontend (Next.js):
    - Client-side rendering to enhance speed and responsiveness.
    - Server-side rendering for SEO and product page preloading.

o

Integration with Sanity CMS for dynamic content like menus and featured products.

2. Backend:

   o REST APIs to manage users, food items, orders, and delivery zones. o Handles business logic, order processing, and integration with external services.

3. Database (MongoDB):

   o NoSQL database to manage flexible and scalable data structures. o Collections for food items, orders, customers, delivery zones, and user authentication.

4. CMS (Sanity):

   o Manages dynamic content like seasonal menus, featured recipes, and blog posts.

5. Order Tracking (ShipEngine):

   o Tracks food deliveries in real-time.

   o Manages shipment and delivery status updates.

6. Authentication (MongoDB):

   o MongoDB securely stores user credentials. o Passwords are encrypted using hashing algorithms (e.g., bcrypt).

7. Deployment:

   Frontend deployed on Vercel. o Backend deployed using AWS Lambda with a serverless architecture.

System Components and Workflow

o

1. User Signup/Login:

   o Input: User credentials (email, password). o Database: MongoDB stores user data with hashed passwords.

   o API Endpoint: POST /register, POST /login, and GET /verifyroute for handling user authentication and verification.

   o Outcome: JWT token issued for session management.

2. Content Management (Sanity CMS):

   o Admin Role: Manages food product listings, seasonal promotions, and blog content. o API Integration: GROQ Queries to fetch dynamic content for the frontend. o Outcome: Content stored and updated in Sanity is rendered on the Next.js frontend.

3. Food Item Browsing and Checkout:

   o Frontend: Next.js provides server-side rendering for product pages. o Database: MongoDB stores food item details (name, price, description, image, etc.).

   API Endpoint: GET /food-items for listing, GET /fooditems/:id for details, and POST /food-items to add items (admin/seller role only). o Outcome: Users browse, add food items to their cart, and proceed to checkout.

4. Order Management:

-
- Database: MongoDB stores order data (customer ID, food item ID, quantity, status). ○ API Endpoint: POST /orders to create new orders (status defaults to "Pending").
- Outcome: Order data processed and stored for tracking.

5. Shipment Tracking (ShipEngine):
   - Integration: ShipEngine API for real-time food delivery tracking. ○ API Endpoint: GET /shipments/:orderId to fetch the delivery status.
   - Outcome: Users receive updates on their order delivery.
6. 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 payment confirmation or COD selection.

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.

Food Item Management

- GET /api/food-items: List all food items.
- GET /api/food-items/:id: Fetch food item details by ID.
- POST /api/food-items: Add a new food item (requires admin/seller role).
- PUT /api/food-items/:id: Update food item details (requires admin/seller role).
- DELETE /api/food-items/:id: Delete a food item (requires admin/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.

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):
  - Handles user interactions and renders data fetched via APIs.
  - Communicates with the backend for authentication, food item data, and order processing.
- Backend APIs:
  - RESTful endpoints for CRUD operations on users, food items, orders, and shipments.
  - Integrated with ShipEngine and multiple payment gateways for third-party functionality.
- Database (MongoDB):
  - Stores user, food item, and order data.
  - Provides scalable and flexible schema designs for rapid iteration.
- Sanity CMS:
  - Manages dynamic content, ensuring marketing and product information is up-to-date.

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.

Food Items

- _id: Unique identifier for the food item.
- name: Name of the food item.
- price: Price of the food item.
- stock: Availability status of the item.
- description: Detailed description of the food item.
- image_url: URL of the food item image.
- user_id: ID of the seller listing the item.

Orders

- order_id: Unique identifier for the order.
- customer_id: Reference to the customer placing the order.
- food_item_id: Reference to the food item ordered.
- quantity: Quantity of the food item ordered.
- status: Current status (e.g., Pending, Confirmed, Completed).
- order_date: Timestamp of when the order was placed.

Timeline.

Day 1-2: Set up Next.js project and configure Sanity CMS

Day 3-4: Build product listing and checkout. 3. Day 5: o Integrate ShipEngine for shipment tracking.

Day 6: Finalize payment gateway integration and test.

Day 7: Deploy and conduct end-to-end testing.
