Marketplace Technical Foundation - General E-Commerce Website

Hackathon Day 2:

Technical Plan for General E-Commerce Website

** Introduction**

This document outlines the comprehensive technical plan for developing a dress shop e-commerce website. The platform aims to provide a seamless user experience for browsing, purchasing, and managing dresses. It leverages modern technologies like Sanity CMS for data management and APIs for real-time functionalities. This plan ensures alignment with the business goals and industry best practices for General E-Commerce marketplaces.

** Business Goals ** -

Provide a user-friendly platform for browsing and purchasing dresses. - Implement real-time inventory updates for accurate product availability. - Ensure secure payment processing and seamless order placement. - Optimize the website for scalability and high performance.

Day-2

1. Technical Requirements

Business Goals & Technical Requirements

1. Frontend (User Interface) - Built Using Next.js

The frontend represents the user-facing aspect of the website, where customers interact with the platform.

- **Responsive Design**: Ensure the website is mobile-friendly, tablet-compatible, and looks great on desktops.
- **User-Friendly**: The website should feature intuitive navigation, making it easy for users to browse dresses, manage their cart, and proceed with checkout.

Essential Pages:

- **Home**: The landing page of the website.
- **Product Listing**: A page displaying products for a specific category.
- **Product Details**: Detailed page for individual products.
- **Cart**: A page displaying the user's cart with product details.
- **Contact**: A page with contact information and a form.
- **About**: Company or website information.
- **FAQS**: Frequently Asked Questions for user reference.
- **Checkout**: A page where users can review their order and input shipping and payment details.
- **Order Confirmation**: A page showing the order details and shipment tracking after checkout.

Homepage Features:

- **Featured Products**: Highlight specific products on the homepage.
- **Promotional Banners**: Display offers or sales.
- **Category Shortcuts**: Links to key categories like Groceries, Electronics, and Fashion.
- **CTAs**: Clear calls-to-action like "Shop Now," "Browse Categories," and "View Deals" to encourage user engagement.

Product Section:

- **Category Pages**: Separate pages for categories like Groceries, Electronics, Fashion, etc.
- **Product Listing Page**: Includes filters (Price, Category, Ratings) and sorting options (Best Sellers, Price, New Arrivals).

- **Product Details Page**: Displays the product title, images, description, price, stock availability, discounts, ratings, reviews, and an FAQ section.
- **Cart Page**: Allows users to review quantities, product prices, and overall totals.
- **Checkout Page**: Collects shipping information and payment details.
- **Order Confirmation Page**: Displays order details and shipment tracking.

---

2. Backend (Sanity CMS)

The backend will handle product data and order management, ensuring smooth data flow between the website and the CMS.

- **Data Management**: Sanity CMS will manage products, customer data, and order records.
- **Schema Design**: Schemas will be created for the following entities:
    - **Products**: Includes fields for product details (e.g., title, price, description).
    - **Orders**: Stores order-specific details such as products ordered, customer, and shipping information.
    - **Customers**: Maintains customer profiles and purchase history.
- **API Integration**: Use **Sanity's APIs** to fetch product, order, and customer data from the backend to the frontend, ensuring real-time updates and seamless user experience.

Sanity Schema Design

Products Schema

Fields:

- **ProductID**: Primary Key
- **Name**: The name of the product.
- **Description**: A detailed description of the product.
- **Category**: The category to which the product belongs.
- **Price**: The cost of the product.
- **Stock Quantity**: The available quantity of the product in stock.
- **Color Options**: The color variants available for the product.
- **Size Options**: The size variants available for the product.
- **Ratings**: Customer ratings for the product.
- **Reviews and FAQs**: Customer reviews and frequently asked questions related to the product.
- **Discount**: Discount applied to the product (if applicable).

Customer Schema

Fields:

- **CustomerID**: Primary Key
- **Full Name**: The full name of the customer.
- **Email**: The email address of the customer.
- **Phone Number**: The contact number of the customer.
- **Address**: The physical address of the customer.
- **Order History**: A record of past orders placed by the customer.
- **Loyalty Points**: Points accumulated by the customer (optional).

Orders Schema

Fields:

- **OrderID**: Primary Key
- **CustomerID**: Foreign Key linking to the Customer Schema.
- **ProductID(s)**: Many-to-Many relationship, representing the products associated with the order.
- **Order Date**: The date when the order was placed.
- **Status**: The current status of the order (e.g., Pending, Shipped, Delivered).
- **Total Amount**: The total price of the order.

Payments Schema

Fields:

- **PaymentID**: Primary Key
- **OrderID**: Foreign Key
- **Amount Paid**: The total amount paid for the order.
- **Payment Method**: The method used for payment (e.g., Credit Card, UPI, Wallet).
- **Payment Status**: The current status of the payment (e.g., Successful, Pending).

Shipment Schema

Fields:

- **ShipmentID**: Primary Key
- **OrderID**: Foreign Key
- **Courier Service**: The service handling the shipment.
- **Tracking Number**: Unique identifier for tracking the shipment.
- **Estimated Delivery Date**: Date when the shipment is expected to arrive.
- **Shipment Status**: Current status of the shipment (e.g., in transit, delive

Implementation Steps:

1. **Design and Test Schemas**: Use **Sanity Studio** to design and test shipment schemas, ensuring they meet all required data points.
2. **Frontend Integration**: Fetch and manipulate data on the frontend using **GROQ queries** for seamless interaction with the schema.
3. **Optimization**: Optimize schemas for scalability and future expansion, ensuring they can handle increased data load and new features.

Third-Party APIs:

To provide critical functionality for the marketplace, the following third-party APIs will be integrated:

Payment Gateways

- **Stripe**
  Features:
  - Secure payment processing.
  - Supports multiple payment methods including credit/debit cards and wallets.
  - Provides real-time transaction updates.
- **Integration**:
  - Use **Stripe SDKs and APIs** to integrate secure payment processing into the website.
- **PayPal**
  Features:
  - Widely accepted payment solution.
  - Supports credit/debit card payments and wallet transactions.
- **Integration**:
  - Use **PayPal's REST API** for handling transactions and ensuring smooth payment processing for customers.

Shipment Tracking APIs

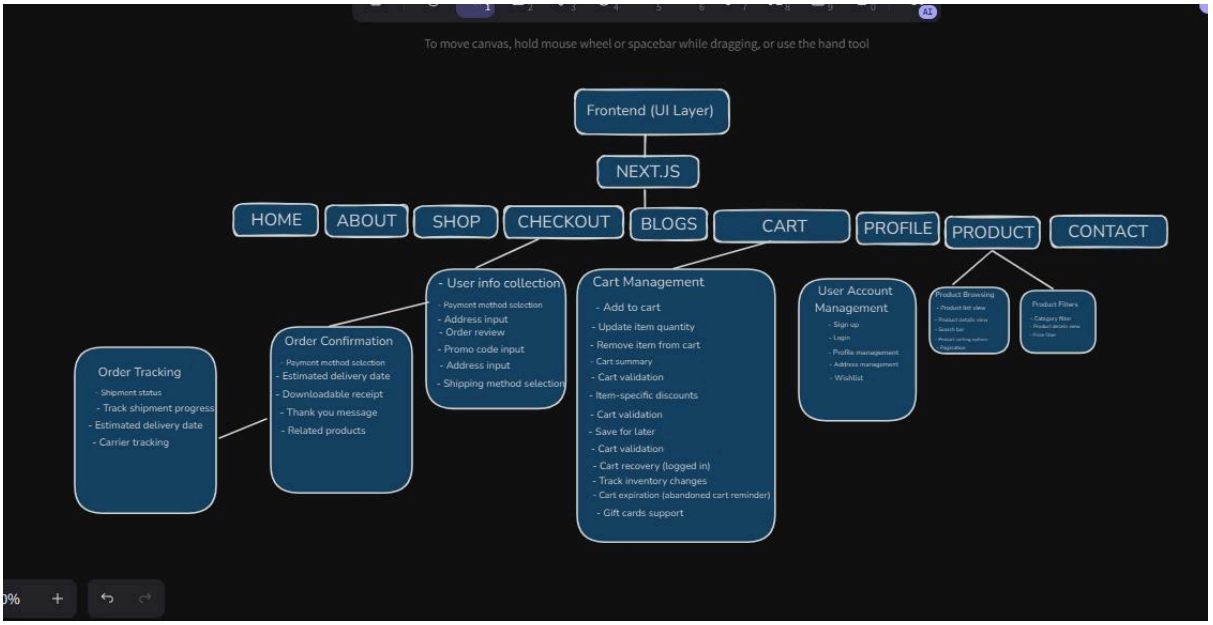
- **ShipEngine**
  Features:
  - Multi-carrier support for different shipment services.
  - Real-time tracking of shipments.
  - Shipping rate comparison to select the best options.
- **Use Case**:
  - Efficient generation of shipment labels and tracking for smooth delivery management.

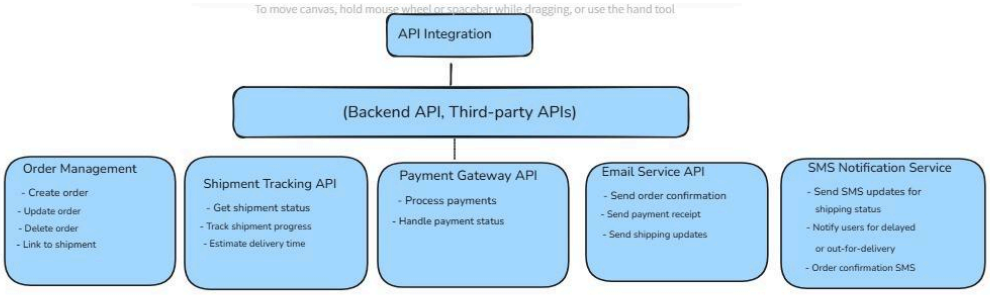
- **AfterShip**
  Features:
    - Real-time shipment tracking.
    - Customer notifications for order updates.
- **Use Case**:
    - Provide live tracking updates to customers, improving the overall delivery experience.
- **EasyPost**
  Features:
    - Shipping label creation.
    - Rate calculation for different carriers.
    - Real-time shipment tracking.
- **Use Case**:
    - Streamline backend logistics, including shipping and delivery management.

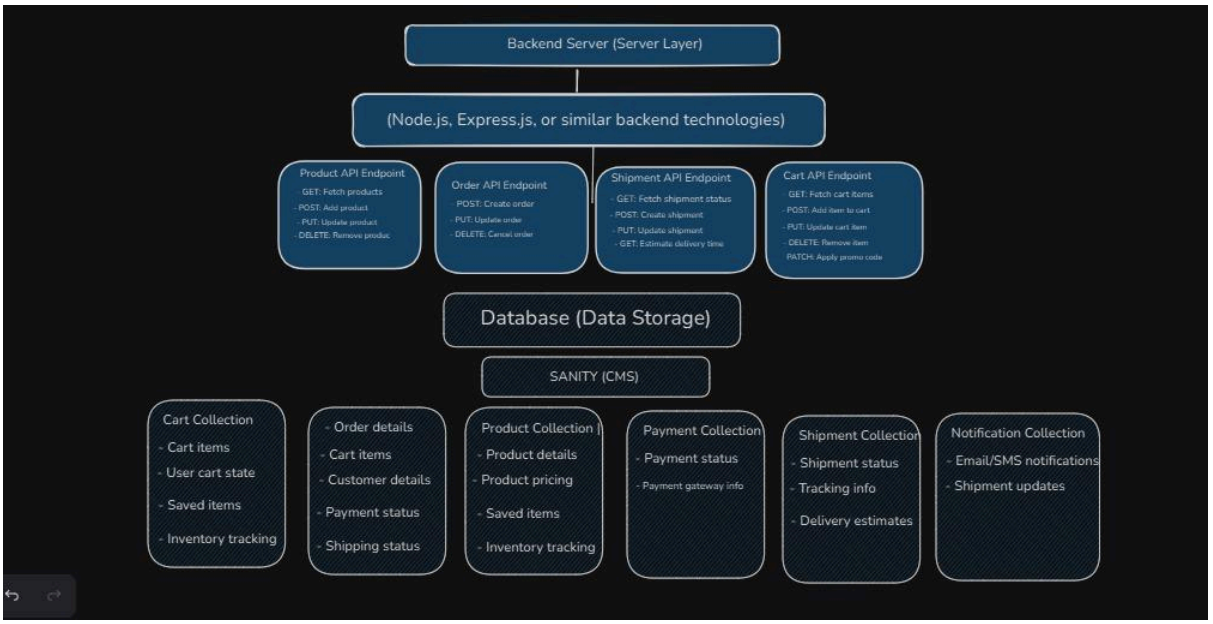
Additional APIs

- **Google Maps API**
  Use Case:
    - Address validation to ensure customer addresses are accurate.
    - Delivery zone mapping to optimize shipping and delivery routes.
- **Notification APIs (Email/SMS)**
  Use Case:
    - Send order confirmations and delivery status updates to customers via **email** and **SMS**, enhancing communication and customer engagement.

2. Design System Architecture







The system architecture is divided into the following key components:

Frontend (Next.js)

- **Role**: Displays the user interface, handles user interactions, and provides server-side rendering for fast load times and SEO optimization.
- **Component Library**: Use **shadcn/ui** for customizable and reusable components.
- **Styling**: Utilize **Tailwind CSS** for a responsive and visually appealing design.

Backend (Sanity CMS)

- **Role**: Manages and stores dynamic content, including products, orders, customer information, delivery zones, and payment data. Provides APIs for the frontend to access and update this data in real-time.

Third-Party APIs

- **Role**: Integrates external services to enhance functionality, including:
    - **Payment Processing** (e.g., **Snipcart**)
    - **Shipping and Tracking** (e.g., **ShipEngine**)
    - Other necessary services to streamline user and order management.

Database (Managed by Sanity)

- **Role**: Stores structured data for products, users, orders, and other dynamic content. Sanity's API enables seamless integration with the frontend for easy access and management.

Deployment

- **Hosting**: Use platforms like **AWS**, **Vercel**, or **Netlify** for deployment to ensure high availability and scalability.
- **CI/CD**: Implement automated deployment pipelines using **GitHub Actions** or **Jenkins** to streamline updates and maintain code quality.

Key Workflows

---

1. User Browsing:

- A user visits the marketplace frontend to browse available products.
- The frontend requests product listings from the **Product Data API** to display on the site.

2. Product Display:

- The **Product Data API** fetches product data from **Sanity CMS**.
- Product details such as name, price, description, and images are dynamically displayed on the site.

3. Add to Cart:

- Users can add selected products to their shopping cart.
- The frontend updates the **cart state** in real-time, reflecting the products the user has chosen.

4. Order Placement:

- Once the user decides to place an order, the order details (product IDs, quantities, etc.) are sent to **Sanity CMS** via an API request.
- The order is recorded in **Sanity CMS** for tracking and future reference.

5. Shipment Tracking:

- Shipment tracking information is fetched through a **Third-Party API** (e.g., **ShipEngine**).
- Real-time shipment tracking updates are displayed to the user, keeping them informed about their order's delivery status.

6. Payment Processing:

- Payment details (including user payment method) are securely processed via a **Payment Gateway** (such as **Stripe** or **PayPal**).
- Upon successful payment, a confirmation is sent back to the user and is recorded in **Sanity CMS** for order tracking.

7. Order Confirmation & Delivery:

- Once payment is confirmed, an **order confirmation page** is generated for the user, providing order details and confirmation.
- A **shipping label** is generated using **ShipEngine**, and **tracking details** are provided to the customer for real-time tracking.
- Users can track their shipments and receive delivery updates through the frontend.

3. Plan API Requirements

API Endpoints Summary for eCommerce Platform

Endpoint Name	Method	Description	Request Body	Response Example

/api/users/register	POST	Registers a new user.	`{ "username": "john", "email": "john@example.com", "password": "pass123" }`	`{ "status": "success", "message": "User registered successfully." }`
/api/users/login	POST	Authenticates a user and generates a JWT.	`{ "email": "john@example.com", "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}/prod	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_method": "card", "shipping_address": { ... } }`	`{ "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_products": [ ... ] } }`

4-Technical Roadmap for eCommerce Platform Development

Phase 1: Planning

- **Objective**: Define the project scope, features, and wireframes.
- **Tasks**:
    - Finalize the project goals and features (e.g., product listing, cart functionality, payment and shipment integration).
    - Create UI/UX wireframes and design mockups.
    - Identify third-party integrations (payment gateways, shipment tracking APIs).
- **Deliverables**:
    - Project Scope Document
    - UI Mockups & Wireframes

Phase 2: Frontend Development

- **Objective**: Set up the frontend with Next.js, integrate shopping cart functionality, and connect with Sanity CMS.
- **Tasks**:
    - Set up Next.js for server-side rendering and dynamic content loading.
    - Implement product catalog display and cart functionality (add/remove products).
    - Fetch product data from Sanity CMS API and display dynamically on the frontend.
- **Deliverables**:
    - Functional product catalog
    - Shopping cart integration
    - Basic frontend UI with real-time cart updates

Phase 3: Backend Development

- **Objective**: Set up backend infrastructure, API routes for product data, orders, and customer management.
- **Tasks**:
    - Set up Sanity CMS to manage dynamic content (product data, customer profiles, orders).
    - Develop API routes for product catalog, cart management, and order processing.
    - Integrate third-party APIs for payment processing (e.g., Snipcart) and shipment tracking (e.g., ShipEngine).
- **Deliverables**:
    - Backend infrastructure (Sanity CMS setup)
    - Functional API routes for product data, orders, and customer management
    - Third-party API integrations (payment and shipment)

Phase 4: Payment & Shipping Integration

- **Objective**: Integrate payment gateway (Snipcart) and shipment tracking (ShipEngine).
- **Tasks**:
    - Integrate Snipcart API to handle secure payments.
    - Integrate ShipEngine API to handle shipment label creation and tracking.
    - Test payment processing and shipment tracking functionalities.
- **Deliverables**:
    - Payment gateway (Snipcart) integration
    - Shipping label creation and tracking (ShipEngine) integration

---

Phase 5: Testing

- **Objective**: Thoroughly test the website for functionality, responsiveness, and security.
- **Tasks**:
    - Conduct functional testing (product browsing, cart updates, checkout process).
    - Test responsiveness across various devices (mobile, tablet, desktop).
    - Perform security testing (user data protection, secure payments).
    - Fix any bugs identified during testing.
- **Deliverables**:
    - Bug-free platform
    - Full system testing (functionality, responsiveness, security)

Phase 6: Deployment

- **Objective**: Deploy the website to a live environment and set up monitoring.
- **Tasks**:
    - Deploy the website using a hosting platform (e.g., Vercel, Netlify).
    - Set up performance monitoring (e.g., Google Analytics, New Relic).
    - Implement error tracking (e.g., Sentry, LogRocket).
- **Deliverables**:
    - Live website
    - Continuous performance and error monitoring setup

Phase 7: Post-Launch

- **Objective**: Monitor the website’s performance and implement regular updates.
- **Tasks**:
    - Monitor website traffic and performance metrics.

    - Address any user feedback and bug reports.
    - Implement regular updates and new features (e.g., promotions, new product categories).
- **Deliverables**:
    - Ongoing site maintenance
    - Feature improvements and bug fixes

5 - . Category-Specific Instructions (Updated)

Product Features by Category

Fashion

- **Size & Color Selection**:
    - Allow users to select different sizes and colors for clothing and accessories.
    - Provide an easy-to-use interface for selecting options, with available stock shown in real-time.
- **Virtual Try-On (AR/VR)**:
    - Implement Augmented Reality (AR) or Virtual Reality (VR) technologies to allow customers to virtually try on clothes, shoes, and accessories.
    - This feature can include models or a "mirror mode" where users can see how items would look on them before making a purchase.

Home Essentials

- **Bundle Offers**:
    - Introduce product bundles (e.g., buying a set of kitchen appliances or furniture) to encourage bulk purchases and increase sales.
    - Offer discounts or special promotions for bundled purchases to make them more appealing.
- **Specifications & Care Instructions**:
    - Provide detailed specifications for home essentials (dimensions, materials, features) to help customers make informed decisions.
    - Include care instructions (cleaning, maintenance) for products to ensure longevity and customer satisfaction.

Health & Wellness

- **Certifications & Lab Test Reports**:

    - Display certifications, lab test results, and third-party testing information to assure customers of the product's quality and safety, especially for supplements, vitamins, and organic products.
    - Make these documents easily accessible, enhancing transparency and trust.
- **Subscriptions for Recurring Orders**:
    - Offer subscription services for recurring health and wellness products (e.g., vitamins, supplements) with automatic deliveries at regular intervals.
    - Provide incentives such as discounts for subscription-based purchases to encourage customer loyalty.

Electronics

- **Product Specifications & Comparison Tools**:
    - Showcase detailed product specs for electronic items (processor speed, storage capacity, battery life, etc.).
    - Integrate a comparison tool that lets users compare products side by side to help them make the best choice based on their needs.
- **Extended Warranty & Service Plans**:
    - Offer extended warranties and service plans as an upsell for electronics, providing customers with peace of mind about their purchases.
    - Include details about the terms, coverage, and any special services included in these plans (e.g., free repairs, priority customer service).

6 - Collaborate and Refine (Updated)

Collaborate and Refine (Updated)

Feedback Integration

- **Continuous Feedback Loop**:
    - Regularly collect feedback from stakeholders (e.g., business owners, marketing teams) and end-users (e.g., customers) to understand their needs and expectations.
    - Use tools like surveys, user interviews, or analytics to gather insights into how users are interacting with the platform and which features need improvement.
    - Prioritize feedback to make informed decisions on feature updates or enhancements.
- **Adapt and Improve**:
    - Implement the most critical feedback first, ensuring that the product evolves according to customer expectations and business goals.
    - Track changes made based on feedback and monitor whether they resolve pain points or improve user experience.

Code Reviews

- **Peer Reviews**:
    - Conduct thorough peer reviews of all code changes to ensure high code quality and prevent issues in the production environment.
    - Involve different team members with various expertise to check for code readability, maintainability, and security risks.
- **Maintain Consistency**:
    - Ensure that the codebase follows coding standards, conventions, and best practices (e.g., clean code, DRY principles).
    - Use automated tools like linters to enforce style guidelines and identify issues early.
- **Issue Identification**:
    - Review code for potential bugs, performance issues, and compatibility problems.
    - Address issues discovered during the review process promptly, ensuring faster identification of problems before deployment.

Iterative Testing

- **Unit Testing**:
    - Develop unit tests for individual components or functions to verify that each piece of code works as expected.
    - Use testing frameworks like Jest for JavaScript/Next.js components and Mocha for API endpoints.
- **Integration Testing**:
    - Implement integration tests to verify that different components and services (e.g., frontend and backend) work together as expected.
    - For example, test the interaction between the frontend and Sanity CMS APIs, or between the cart functionality and the checkout process.
- **UI Testing**:
    - Use tools like Cypress or Selenium to test the user interface and ensure that the application behaves as intended in real-world scenarios.
    - Include tests for responsiveness, interaction elements (like buttons, forms), and cross-browser compatibility.
- **Automated Testing Pipeline**:
    - Integrate automated testing into the Continuous Integration (CI) pipeline to ensure code changes are automatically tested and validated before deployment.

Documentation Updates

- **Update Architecture & Workflows**:

    - Regularly update system architecture diagrams, technical workflows, and data flow documentation to reflect changes in the platform’s features or infrastructure.
    - Keep track of any architectural changes (e.g., migration to a new CMS, integration with additional third-party APIs) for clarity among team members.
- **API Documentation**:
    - Ensure that API documentation is kept up-to-date with new endpoints, request parameters, and response examples.
    - Use tools like Swagger to generate interactive API documentation, allowing developers to easily explore the API.
- **User & Developer Guides**:
    - Maintain detailed user guides for customers and admin users, explaining how to use the platform's key features.
    - Ensure developer documentation includes setup instructions, dependency management, and guidelines for contributing to the codebase.

THANK YOU

Presentation By :

Humaiza Naz & Ramesha Javed

00080465 00070760

Sunday 2-5 Sunday 2-5