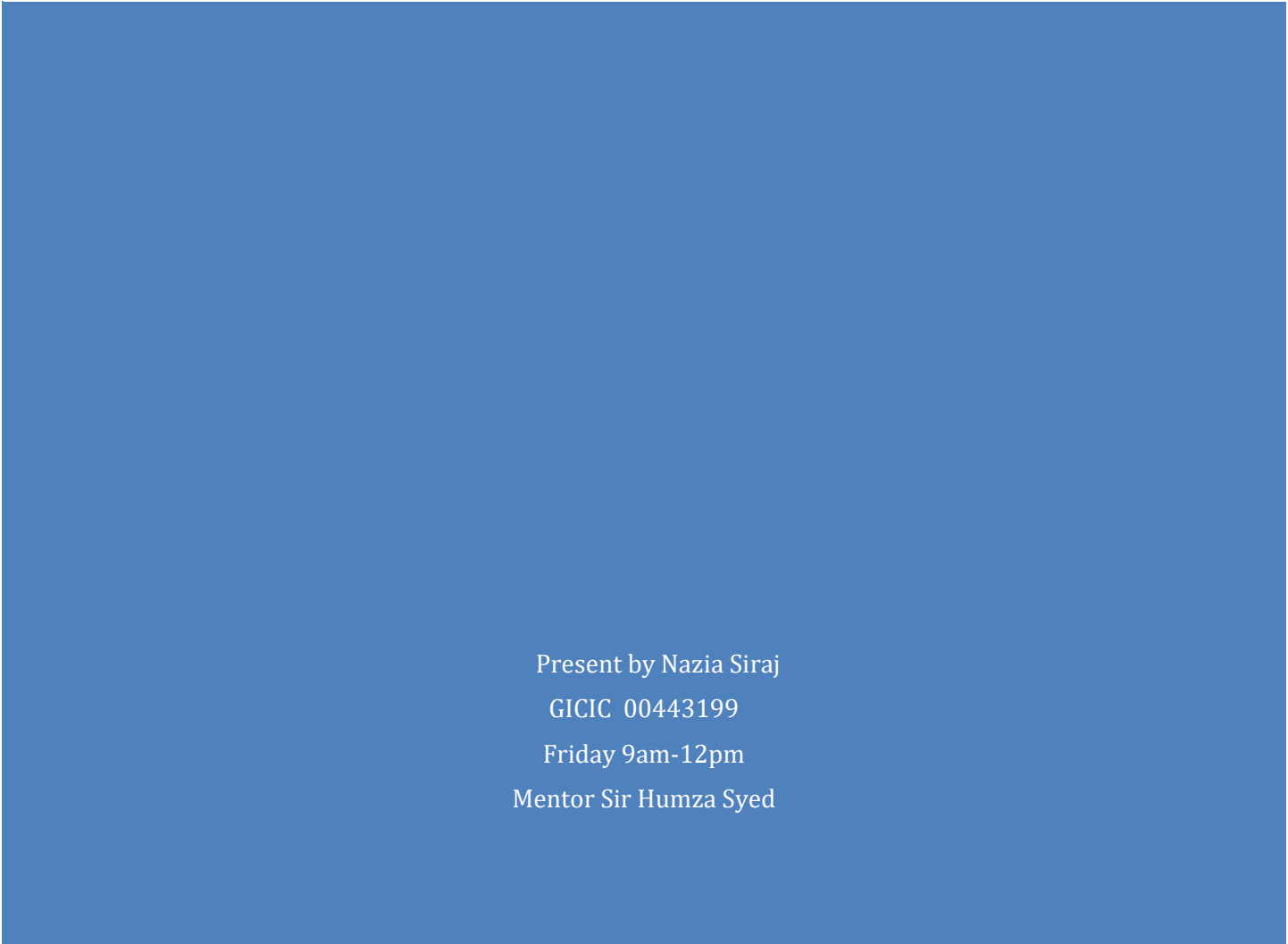


# DAY-2 TECHNICAL PLAN FOR RESTAURANT Q-COMMERCE “FOOFTUCK” WEBSITE



Present by Nazia Siraj  
GICIC 00443199  
Friday 9am-12pm  
Mentor Sir Humza Syed

# **Marketplace Technical Foundation - Q-Commerce**

## **“FoodTuck” Restaurant Website:**

**\*\* Hackathon Day 2: \*\***

### ***Technical Plan for Restaurant Q-Commerce “FoofTuck” Website***

#### **\*\*Introduction\*\***

This document outlines the comprehensive technical plan for developing a restaurant Q-commerce website. The platform aims to provide a seamless user experience for browsing menus, placing orders, and managing deliveries. 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 Q-commerce restaurant marketplaces.

#### **\*\*Business Goals\*\***

Provide a user-friendly platform for browsing restaurant menus and placing orders.

Implement real-time updates for menu availability and order tracking.

Ensure secure payment processing and a seamless checkout experience.

Optimize the website for scalability, high performance, and smooth delivery operations.

## Day-2

### 1. Technical Requirements

#### Business Goals & Technical Requirements for Q-Commerce

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

The frontend must focus on speed, efficiency, and a seamless user experience.

- **Responsive Design:** Ensure the website is optimized for mobile users, as quick commerce heavily relies on mobile interactions.
- **User-Friendly Navigation:** Simplify navigation to allow users to find products, manage their cart, and complete purchases in the shortest possible time.

##### Essential Pages:

- **Home:** Highlight quick deals and express delivery products.
- **Product Listing:** Showcase products with tags for delivery time (e.g., "Delivered in 30 minutes").
- **Product Details:** Display delivery time, stock availability, and product information clearly.
- **Cart:** Provide real-time updates on delivery charges and estimated delivery time.
- **Contact:** Include a support chat feature for immediate assistance.
- **About:** Highlight the Q-commerce mission, focusing on speed and convenience.
- **FAQs:** Address questions about delivery times, returns, and payments.
- **Checkout:** Ensure a fast, streamlined checkout process with saved addresses and payment methods.
- **Order Confirmation:** Provide live tracking details and estimated delivery time updates.

##### Homepage Features:

- **Featured Products:** Highlight items with express delivery options.
- **Promotional Banners:** Focus on limited-time offers, express delivery discounts, and flash sales.
- **Category Shortcuts:** Emphasize daily essentials (e.g., Groceries, Snacks, Pharmacy) for quick access.
- **CTAs:** Use clear call-to-actions like "Get It in 30 Minutes!" or "Order Now for Instant Delivery."

### Product Section:

- **Category Pages:** Focus on high-demand categories like groceries, fresh produce, medicines, and ready-to-eat meals.
- **Product Listing Page:** Include delivery filters (e.g., "Delivered in under 30 minutes") and sort options for availability.  
Includes filters (e.g., Cuisine Type, Price Range, Dietary Preferences, Ratings).

Sorting options (e.g., Best Sellers, Low to High Price, New Arrivals).

- **Product Details Page:**

Displays the dish name, images, description, price, availability, dietary information (e.g., Vegan/Gluten-Free), customer ratings, reviews, and FAQs (e.g., portion size, ingredients).

- **Cart Page:**

Allows users to review selected dishes, customize quantities, and view the total price.

- **Checkout Page:**

Collects delivery address, special instructions, and payment details.

- **Order Confirmation Page:**

Displays order details, estimated delivery time, and tracking information

## 2.Backend (Sanity CMS)

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

### 1. Data Management:

Sanity CMS will manage dishes, customer profiles, and order records.

### 2. Schema Design:

Schema will be created for the following entities:

- **Dishes:** Includes fields for menu items (e.g., dish name, description, price, ingredients).
- **Orders:** Stores specific details about orders such as dishes ordered, customer details, and delivery address.
- **Customers:** Maintains customer profiles, order history, and preferences.

### 3. API Integration:

**Use Sanity's APIs:** to fetch menu items, orders, and customer data from the backend to the frontend, ensuring real-time updates for a seamless user experience.

#### Sanity Schema Design:

##### Dishes Schema:

##### Fields:

- **DishID:** Primary Key
- **Name:** The name of the dish.
- **Description:** A detailed description of the dish.
- **Category:** The category (e.g., Appetizers, Beverages) to which the dish belongs.
- **Price:** The cost of the dish.
- **Stock Quantity:** The number of servings available.
- **Dietary Info:** Indicates if the dish is Vegan, Gluten-Free, etc.
- **Ingredients:** Key ingredients used in the dish.

- **Ratings:** Customer ratings for the dish.
- **Reviews and FAQs:** Customer reviews and frequently asked questions related to the dish.
- **Discount:** Discount applied to the dish (if applicable).

#### **Orders Schema:**

##### **Fields:**

- **OrderID:** Primary Key
- **CustomerID:** Links to the customer who placed the order.
- **DishIDs:** List of dishes included in the order.
- **Quantity:** Number of servings per dish.
- **TotalPrice:** Total cost of the order.
- **DeliveryAddress:** Address for order delivery.
- **SpecialInstructions:** Notes for the kitchen/delivery staff.
- **Status:** Current order status (e.g., Pending, In Progress, Delivered).

#### **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 location for rapid delivery, optimized for real-time delivery zones.
- **Order History:** A record of past orders placed by the customer, emphasizing quick delivery patterns.
- **Loyalty Points:** Points accumulated by the customer (optional).
- **Preferred Delivery Time:** A field to capture customers' preferred delivery times for convenience.

### Orders Schema:

#### Fields:

- **OrderID:** Primary Key
- **CustomerID:** Foreign Key linking to the Customer Schema.
- **ProductID(s):** Many-to-Many relationship, representing the products in the order.
- **Order Date:** The date and time when the order was placed.
- **Status:** The current status of the order (e.g., Preparing, Out for Delivery, Delivered).
- **Total Amount:** The total price of the order.
- **Delivery Window:** A short time frame within which the order is promised to be delivered (e.g., 15-30 minutes).
- **Special Instructions:** Optional field for customer instructions, like "Ring the bell" or "Leave at the door."

### 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, Cash on Delivery).
- **Payment Status:** The current status of the payment (e.g., Successful, Pending).
- **Instant Refund Eligibility:** Specifies whether the payment qualifies for instant refunds in case of cancellation.

### Shipment Schema:

#### Fields:

- **ShipmentID:** Primary Key
- **OrderID:** Foreign Key
- **Courier Service:** Service or delivery personnel assigned for the order.
- **Delivery Agent Name:** The name of the delivery agent (optional for tracking).
- **Real-Time Location:** The live location of the delivery agent or shipment.
- **Tracking Number:** Unique identifier for tracking the shipment (optional for small-scale deliveries).
- **Estimated Delivery Time:** The dynamically calculated time of arrival (e.g., "15 minutes").
- **Shipment Status:** Current status of the shipment (e.g., Preparing, Out for Delivery, Delivered)

### Additional Modifications :

#### 1. Real-Time Inventory Management:

- Maintain a real-time inventory database to show only products available for instant delivery.

#### 2. Product Schema Adjustments:



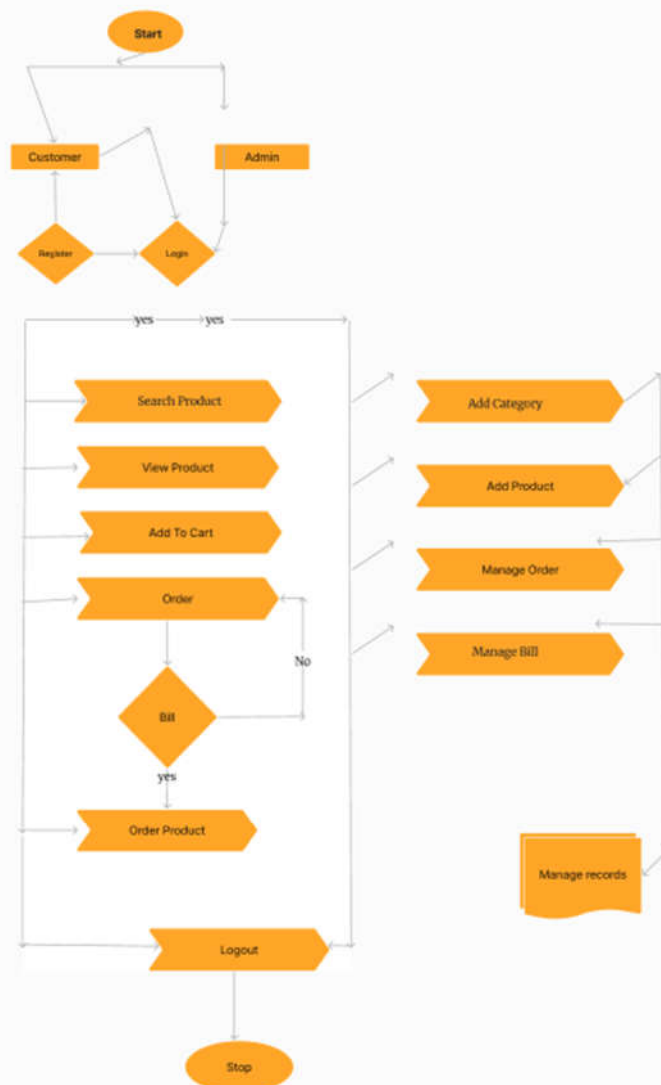
- Add a field for "Delivery Time" to prioritize products that can be delivered within the Q-commerce time frame.

### **3. Delivery Hub Schema (Optional for scaling):**

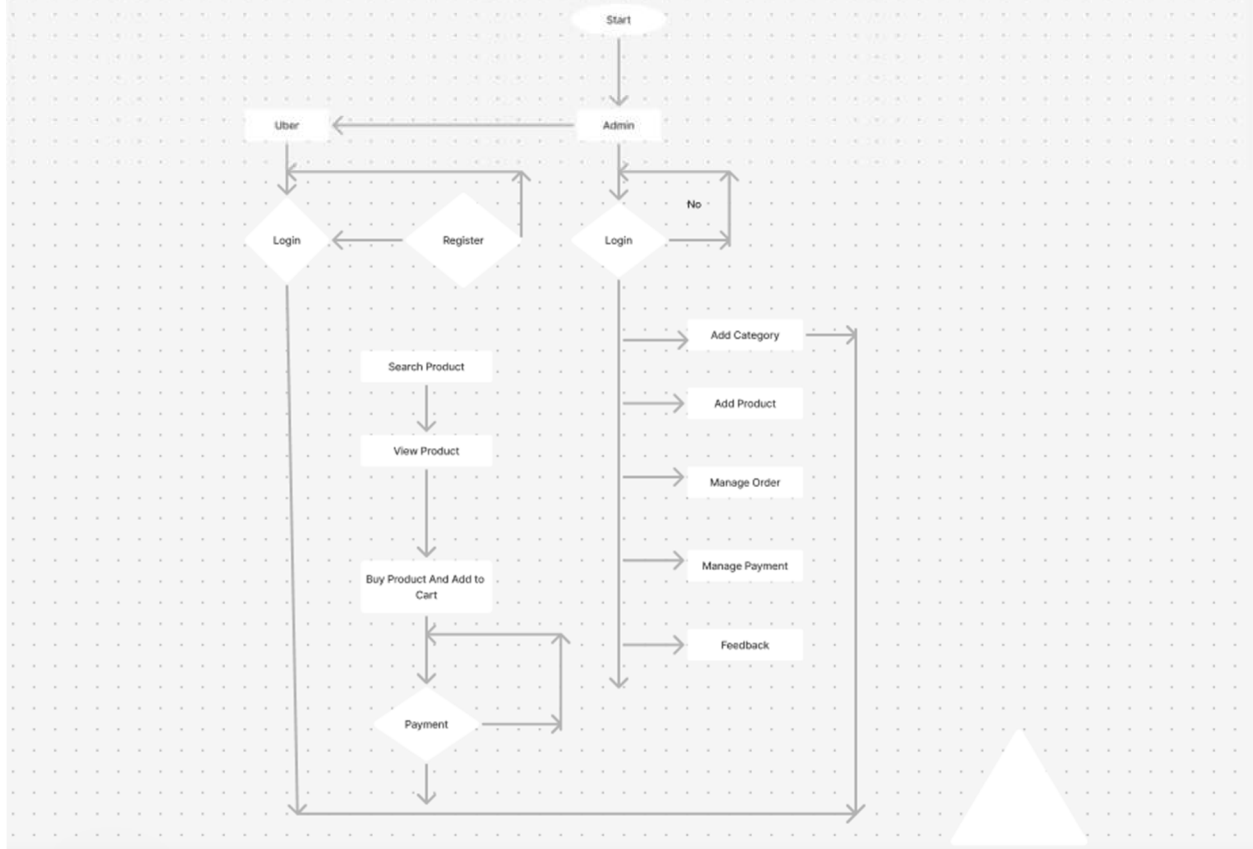
- Include a Delivery Hub schema for storing information about hubs or dark stores that fulfill rapid deliveries.

## FlowChart -1

Frontend requirement: Flow Chart By Nazia Siraj



## Frontend Requirements Flow Chart By Nazia Siraj



### Third-Party APIs :

#### 1. Aftership (Real-Time Shipment Tracking)

##### Features:

- **Real-time shipment tracking:** For providing accurate, up-to-the-minute delivery updates.
- **Customer notifications:** Real-time alerts on order progress (e.g., "Out for delivery," "15 minutes away," etc.).
- **Use Case :**
- **Live tracking updates:** Show customers their order status in real-time. Include features like live location tracking of delivery agents to set precise expectations (e.g., "Your order will be at your doorstep in 10 minutes").

- **Instant delivery updates:** Trigger notifications for quick order status changes (e.g., "Your order has been dispatched" or "Order delivered") with ultra-low latency for customer convenience.

---

## 2. EasyPost (Shipping and Delivery Management):

### Features:

- **Shipping label creation:** Simplifies the process of generating labels for deliveries.
- **Rate calculation:** Provides shipping rate comparisons between multiple carriers for optimal choices.
- **Real-time shipment tracking:** Ensure all customers can track their shipments in real-time.
- **Use Case :**

**Streamlined logistics:** Use local courier services (e.g., Postmates, Uber Direct) to optimize for fast deliveries within a 15–30 minute window.

**Dynamic shipping rate calculation:** Dynamically calculate delivery costs based on the distance to the customer, the delivery time slot, and any traffic factors.

**Instant label creation:** Immediately print shipping labels as soon as an order is placed, ensuring prompt dispatch to fulfill the Q-commerce model.

**Efficient backend management:** Automatically assign the nearest available delivery agent to the order based on real-time logistics data.

## 3. Google Maps API (Location and Route Optimization):

### Features:

- **Address validation:** Ensures that addresses entered by customers are valid and complete, reducing delivery issues.
- **Delivery zone mapping:** Helps identify the quickest routes and ensures that products are delivered within the promised time window.
- **Use Case for Q-commerce:**
- **Address accuracy:** Use Google Maps API to ensure that delivery addresses are correct and fall within serviceable areas to enable ultra-fast deliveries.
- **Dynamic route optimization:** Optimize delivery routes in real-time, factoring in traffic conditions, to reduce delivery times (e.g., "10 minutes left for delivery").
- **Real-time geolocation:** Track the delivery agent in real-time, enhancing the delivery transparency for customers, showing them exactly where their order is during the last-mile delivery.

## 4. Notification APIs (Email/SMS)

### Use Case :

- **Instant customer communication:** Send SMS and email alerts for every key order milestone (e.g., "Order confirmed," "Preparing your order," "On its way," "Delivered").
- **Real-time delivery updates:** Notify customers of live tracking updates via SMS/email (e.g., "Your order is 5 minutes away," "Your order has been delivered").
- **Feedback loops:** Trigger a quick customer satisfaction survey immediately after the order is delivered, helping gather feedback to improve Q-commerce operations.

---

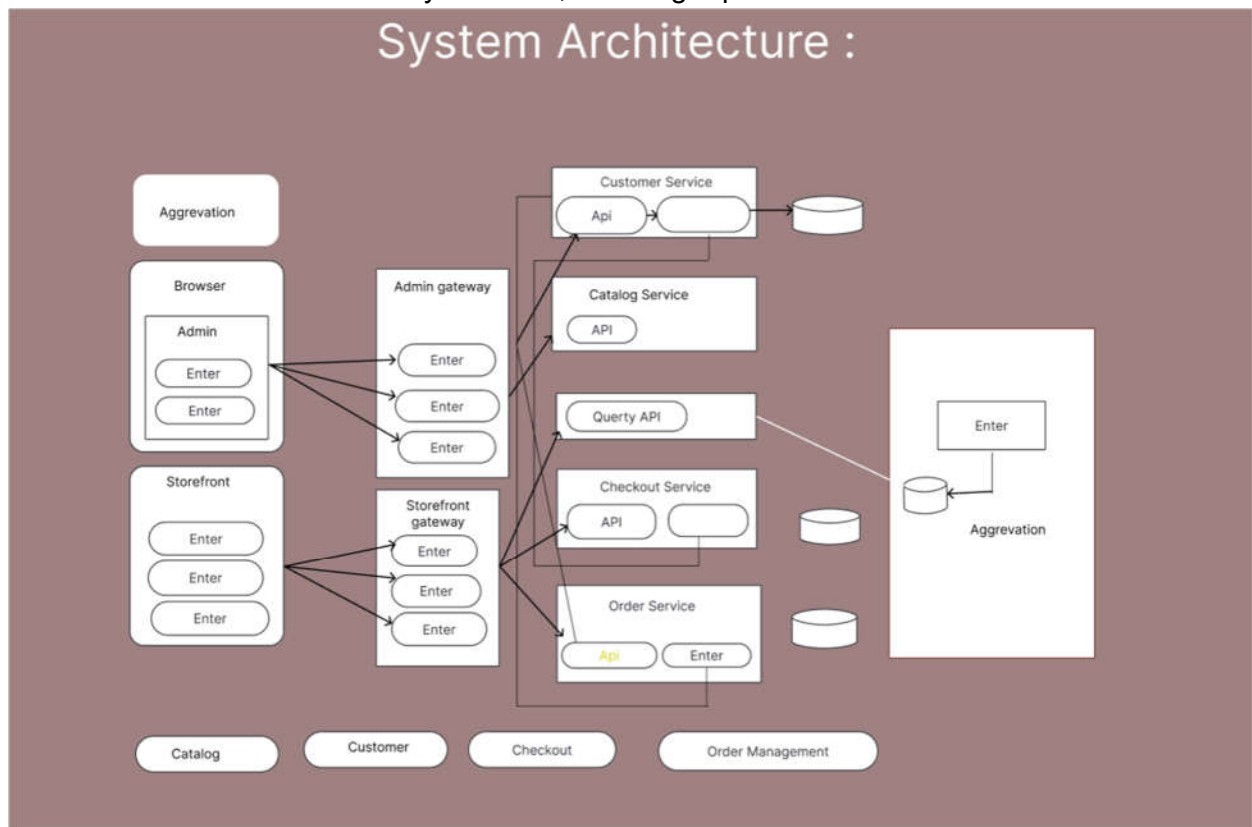
## Key Enhancements for APIs:

**1. Speed and Precision:** All APIs should prioritize real-time accuracy, ensuring that customers can track their deliveries and receive updates instantly.

**2. Dynamic Delivery Management:** Integrate real-time geolocation and delivery time predictions so customers can see live tracking with estimated arrival times.

**3. Mobile-First:** Make sure all notifications (SMS, Email, and App Push) are optimized for mobile use, given that Q-commerce is mostly accessed through mobile apps.

**4. Hyperlocal Optimization:** Adjust route optimization and address validation to suit micro-fulfillment and dark store delivery methods, ensuring rapid order fulfillment.



Frontend (Next.js)

**Role:**

Displays the user interface, handles user interactions, and ensures extremely fast load times and SEO optimization for Q-commerce.

Focus on instant product availability, real-time tracking, and order updates to provide customers with immediate access to products that can be delivered within minutes.

### **Component Library:**

Shadcn/ui for customizable and reusable components, focusing on quick updates for product listings, delivery time windows, and order status.

### **Styling:**

Tailwind CSS to ensure a responsive and mobile-first design, especially important for Q-commerce, where users will interact with the site through mobile devices.

### **Key Modifications:**

Include real-time notifications (e.g., "Your order is being prepared," "Your order is on its way," "5 minutes remaining").

Quick checkout flows to reduce friction, enabling instant payments through wallets or fast payment methods.

Show estimated delivery windows dynamically (e.g., "Delivered in 20 minutes").

---

### **Backend (Sanity CMS)**

#### **Role:**

Manages and stores real-time data such as products, orders, customers, delivery zones, and payment data.

Provides APIs for the frontend to access and update data instantly, focusing on rapid order processing and delivery updates.

### **Key Modifications for Q-commerce:**

Store real-time inventory levels for products that can be delivered instantly.

Delivery time slots and order status must be updated in real-time for transparency.

Customer geolocation data can help optimize delivery routes and ensure the fastest deliveries.

---

### **Third-Party APIs:**

#### **Role:**

Integrates external services to enhance Q-commerce functionality, including:

#### **1. Payment Processing (e.g., Stripe, PayPal):**

Fast payment processing for instant checkouts, especially for mobile wallets and one-click payment solutions.

#### **2. Shipping and Tracking (e.g., ShipEngine, Aftership):**

Real-time shipment tracking and live notifications to ensure customers are always aware of their order status.

Multi-carrier support for hyperlocal delivery services like Postmates, Uber Direct, or local courier companies.

#### **3. Notification Services:**

SMS and email alerts for real-time order updates and confirmations, ensuring constant communication with the customer.



## **Key Modifications :**

Integrate APIs like Uber Direct or Roadie for last-mile delivery, ensuring customers get their products within a short time frame.

Instant refund processing for order cancellations, and support real-time order updates.

---

## **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 to and management of real-time data.

## **Key Modifications for Q-commerce:**

Implement real-time updates for inventory, order status, and delivery times

Use real-time data streams to reflect changes like product availability or order processing status.

---

## **Deployment**

- **Hosting:**

Use platforms like Vercel, AWS, or Netlify to ensure high availability and low-latency deployment that supports instant updates and real-time interactions.

- **CI/CD:**

Implement automated pipelines using GitHub Actions or Jenkins to streamline rapid updates and maintain code quality, especially for Q-commerce's need for constant iteration and changes.

## Key Modifications for Q-commerce:

Ensure high scalability to handle sudden bursts in order volume and to support real-time delivery tracking features.

Fast rollback mechanisms in case of service failures during peak times to maintain customer satisfaction.

---

## Key Workflows for Q-commerce:



### 1. User Browsing:

A user visits the marketplace and sees instant product availability with real-time delivery estimates.

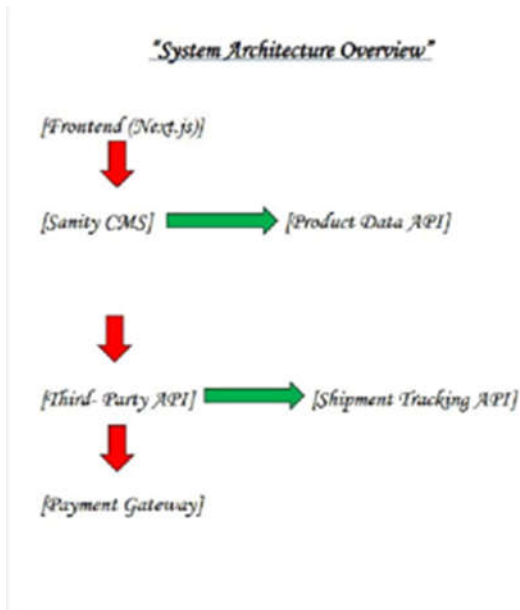
The frontend queries the Product Data API for available products that are ready for immediate delivery.

Users can see product details along with live availability and delivery time slots.

### 2. Product Display:

Display only those products that are available for quick delivery (within 30 minutes or less).

Each product has a visible estimated delivery window (e.g., “Delivered in 20 minutes”) to set expectations.



### 3. Order Placement:

Users place orders for products with clear expected delivery times.

Once an order is confirmed, real-time notifications inform the customer of the order status (e.g., “Your order is being packed,” “Your order is on its way”).

### 4. Checkout & Payment:

Fast checkout options are enabled for mobile wallets, credit/debit cards, or instant payment methods to streamline the buying process.



Once payment is confirmed, a real-time order update (e.g., "Your order is confirmed, preparing for delivery") is sent.

## **5. Real-Time Delivery & Tracking:**

As the delivery progresses, customers receive real-time tracking updates via SMS/email and can monitor the location of the delivery agent.

The estimated delivery window is adjusted dynamically based on real-time traffic conditions and order fulfillment.

### **1. Product Data API (Fetch Real-Time Product Data)**

**Description:** The Product Data API still fetches product data from Sanity CMS, but for Q-commerce, the product details will need to be updated more frequently to reflect availability in real-time. Products need to be in stock locally (hyperlocal) for immediate dispatch.

#### **Example Modifications:**

The API should include stock availability for hyperlocal areas.

Consider adding a real-time "Ready to Dispatch" or "Available in Nearby Stores" status.

### **2. Add to Cart (Real-Time Cart Updates)**

**Description:** Users can add products to their cart with a sense of urgency due to the focus on speed. The cart should be synchronized with a real-time inventory system.

#### **Example Modifications:**

**Local Inventory Check:** When a user adds an item to the cart, check if the product is available for fast delivery in their local area.

**Cart Timer:** A countdown or estimated delivery time should be displayed to indicate when the user can expect the item.

### 3. Order Placement (Hyperlocal Dispatch)

**Description:** Orders should be processed for immediate dispatch. Once the user decides to place an order, the API should handle real-time routing to a local fulfillment center.

#### **Example Modifications:**

Ensure the order details (e.g., product IDs, quantities) are sent to a nearby warehouse/store for instant dispatch.

**Real-Time Dispatch:** Add functionality for real-time updates on the availability and readiness for dispatch from local stores.

### 4. Shipment Tracking (Instantaneous Delivery Tracking)

**Description:** For Q-commerce, shipment tracking must be near-instantaneous. The API should fetch real-time delivery updates for immediate tracking.

#### **Example Modifications:**

Use services like Uber Eats, DoorDash, or hyperlocal courier services that offer near-instant delivery tracking.

**Real-Time Dispatch Updates:** Send updates directly to the user when the delivery is out for dispatch and nearing their location.

### 5. Payment Processing (Seamless Quick Payment)

**Description:** Payments should be processed instantly to match the Q-commerce model. Consider integrating quick payment options.

#### **Example Modifications:**

**Quick Pay Options:** Integrate services like Google Pay, Apple Pay, or other mobile wallets for faster transactions.

**Instant Payment Confirmation:** Payment should be processed and confirmed instantly, with an immediate acknowledgment sent back to the user.

## **6. Order Confirmation & Delivery (Fast Confirmation and Local Delivery)**

**Description:** Order confirmation and delivery status need to be as immediate as possible in a Q-commerce setting. After payment confirmation, the order is fulfilled from a nearby local store or warehouse for instant delivery.

### **Example Modifications:**

**Instant Order Confirmation:** Upon successful payment, provide the user with immediate confirmation and expected delivery times.

**Local Courier Integration:** Generate the shipping label via local courier services or partners to provide faster shipping times.

**Real-Time Delivery Updates:** Provide the user with live tracking of their delivery, indicating exact delivery time windows and real-time tracking.

## **7. Plan API Requirements (Updated for Q-commerce)**

API Endpoints Summary for Q-commerce Platform

Endpoints	Method	Description	Parameters	Response Example
/api foods	GET	Fetch all food items	Name	{id: 1,name:"Pizza"}
/api foods/id	GET	Fetch a single food item	id(Path)	{id: 1,name:"Pizza"}
/api foods	POST	Add a new food item	name, price, category (Body)	{success : true ,id:5}
/api foods/id	PUT	Update a food item	id(Path),name, price(Body)	{success : true }
/api foods/id	DELETE	Delete a food item	id(Path)	{success : true }
/api /categories	GET	Fetch all food categorgies	None	{categories: ["Drinks"]}

Endpoint	Method	Description	Request Body	Response
/api/customers/register	POST	Registers a new customer.	{ "username": "john", "email": "john@example.com", "password": "pass123" }	{ "status": "success", "message": "Customer registered successfully." }
/api/customers/login	POST	Authenticate a customer and generates a JWT.	{ "email": "john@example.com", "password": "pass123" }	{ "status": "success", "token": "jwt.token.here" }

<code>/api/customers/{id}</code>	PUT	Updates customer details.	<pre>{ "username": "john_updated",   "email": "updated@example.com" }</pre>	<pre>{ "status": "success",   "message": "Profile updated successfully." }</pre>
<code>/api/q-categories</code>	GET	Retrieves a list of all food categories.	None	<pre>{ "status": "success",   "data": [ {     "id": 1,     "name": "Beverages"   }, { "id": 2,     "name": "Desserts" } ]</pre>
<code>/api/q-categories/{id}/items</code>	GET	Fetches menu items within a specific category.	None	<pre>{ "status": "success",   "data": [ {     "id": 101,     "name": "Coke",     "price": "1.50" }, {     "id": 102,     "name": "Lemonade",     "price": "2.00" } ] }</pre>
<code>/api/items/{id}</code>	GET	Retrieves details of a specific menu item.	None	<pre>{ "status": "success",   "data": {     "id": 101,     "name": "Coke",</pre>



				<pre> "price": "1.50", "availability ": "in stock" } } </pre>
<code>/api/cart/add</code>	POST	Adds an item to the customer's cart.	<pre> { "item_id": 101, "quantity": 2 } </pre>	<pre> { "status": "success", "message": "Item added to cart." } </pre>
<code>/api/cart</code>	GET	Retrieves the current state of the customer's cart.	None	<pre> { "status": "success", "data": { "items": [ { "id": 101, "name": "Coke", "quantity": 2 } ] } } </pre>
<code>/api/checkout</code>	POST	Processes payment and places an order.	<pre> { "payment_method" : "card", "delivery_addresses": { "street": "123 Main St", "city": "Karachi", "zip": "74000" } } </pre>	<pre> { "status": "success", "message": "Order placed successfully. " } </pre>

<code>/api/orders/{id}</code>	GET	Retrieves the details of a specific order.	None	<pre>{ "status": "success", "data": { "order_id": 12345, "status": "delivered", "items": [ { "id": 101, "name": "Coke", "quantity": 2 } ] } }</pre>
<code>/api/homepage</code>	GET	Retrieves homepage content, including featured menu items.	None	<pre>{ "status": "success", "data": { "featured_items": [ { "id": 201, "name": "Pizza", "price": "10.00" }, { "id": 202, "name": "Pasta", "price": "8.50" } ] } }</pre>

---

Endpoint	Method	Description	Request Body	Response
<code>/api/customers/register</code>	POST	Registers a new customer.	<pre>{ "username": "john", "email": "john@example.co</pre>	<pre>{ "status": "success", "message": "Customer</pre>

			<pre>       "email": "john@example.com",       "password": "pass123"     }   } </pre>	<pre>     {       "status": "registered successfully.",       "token": "jwt.token.here"     } </pre>
/api/customers/login	POST	Authenticates a customer and generates a JWT.	<pre>     {       "email": "john@example.com",       "password": "pass123"     }   } </pre>	<pre>     {       "status": "success",       "token": "jwt.token.here"     } </pre>
/api/customers/{id}	PUT	Updates customer details.	<pre>     {       "username": "john_updated",       "email": "updated@example.com"     }   } </pre>	<pre>     {       "status": "success",       "message": "Profile updated successfully."     } </pre>
/api/q-categories	GET	Retrieves a list of all food categories.	None	<pre>     {       "status": "success",       "data": [         {           "id": 1,           "name": "Beverages"         },         {           "id": 2,           "name": "Desserts"         }       ]     } </pre>
/api/q-categories/{id}/items	GET	Fetches menu items within a specific category.	None	<pre>     {       "status": "success",       "data": [         {           "id": 101,           "name": "Coke",           "price": "1.50"         },         {           "id": 102, </pre>

				<pre>"name": "Lemonade", "price": "2.00" } ] }</pre>
<code>/api/items/{id}</code>	GET	Retrieves details of a specific menu item.	None	<pre>{ "status": "success", "data": { "id": 101, "name": "Coke", "price": "1.50", "availability ": "in stock" } }</pre>
<code>/api/cart/add</code>	POST	Adds an item to the customer's cart.	<pre>{ "item_id": 101, "quantity": 2 }</pre>	<pre>{ "status": "success", "message": "Item added to cart." }</pre>
<code>/api/cart</code>	GET	Retrieves the current state of the customer's cart.	None	<pre>{ "status": "success", "data": { "items": [ { "id": 101, "name": "Coke", "quantity": 2 } ] } }</pre>

/api/checkout	POST	Processes payment and places an order.	{ "payment_method": "card", "delivery_addresses": { "street": "123 Main St", "city": "Karachi", "zip": "74000" } }	{ "status": "success", "message": "Order placed successfully." }
/api/orders/{id}	GET	Retrieves the details of a specific order.	None	{ "status": "success", "data": { "order_id": 12345, "status": "delivered", "items": [ { "id": 101, "name": "Coke", "quantity": 2 } ] } }
/api/homepage	GET	Retrieves homepage content, including featured menu items.	None	{ "status": "success", "data": { "featured_items": [ { "id": 201, "name": "Pizza", "price": "10.00" }, { "id": 202, "name": "Pasta", "price":

```
"8.50" } ] }  
}
```

---

## 4. Technical Roadmap for Q-Commerce Platform Development

### 1. Phase 1: Core Functionality Development

Build customer authentication (register/login with JWT).

Develop cart management APIs.

Create item listing APIs (by category, search, etc.).

### 2. Phase 2: Real-Time Features

Enable real-time inventory updates.

Implement a recommendation engine for featured/related items.

### 3. Phase 3: Checkout and Delivery

Integrate payment gateways (card, wallet, COD).

Develop delivery tracking and quick-order placement.

### 4. Phase 4: Optimization

Add performance enhancements (cache popular items).

Optimize API response time for low-latency operations.

## 5. Phase 5: Scaling

Introduce support for multiple warehouses.

Implement multi-region support for faster delivery.

**Objective:** Define the project scope, features, and wireframes, emphasizing speed and instant delivery.

### Tasks:

Finalize the project goals and features (e.g., item listing, real-time cart updates, fast checkout, delivery tracking).

Create UI/UX wireframes tailored to quick navigation and one-click ordering.

Identify third-party integrations (instant payment gateways, real-time delivery tracking APIs).

### Deliverables:

Q-Commerce Project Scope Document

UI Mockups & Wireframes focusing on speed and simplicity

---

## Phase 2: Frontend Development:

---

Phase 4: Payment & Real-Time Delivery Integration

**Objective:** Integrate instant payment gateways and real-time delivery tracking APIs.

### Tasks:

Integrate a fast and secure payment gateway (e.g., Stripe, Razorpay) to handle instant payments.

Integrate delivery tracking APIs (e.g., Shippo, Mapbox) for real-time delivery updates.

Test payment processing and real-time delivery functionalities to ensure reliability.

**Deliverables:**

Payment gateway integration for instant transactions

Real-time delivery tracking and updates integration

**Phase 5: Testing**

**Objective:** Test the platform thoroughly for instant operations, responsiveness, and security tailored to Q-commerce needs.

**Tasks:**

Conduct functional testing for quick item browsing, real-time cart updates, and fast checkout.

Test responsiveness across various devices (mobile-first approach).

Perform security testing, ensuring secure customer data and payment transactions.

Identify and resolve any bugs or slow responses during testing.

**Deliverables:**

Bug-free, fast, and secure Q-commerce platform

Comprehensive testing reports (functionality, responsiveness, and security)



---

## **Phase 6: Deployment**

**Objective:** Deploy the Q-commerce platform to a live environment optimized for speed and scalability.

### **Tasks:**

Deploy the platform using a scalable hosting solution (e.g., Vercel, Netlify, or AWS).

Set up performance monitoring tools (e.g., Google Analytics, New Relic) to track speed and traffic.

Implement error tracking tools (e.g., Sentry, LogRocket) for instant issue detection.

### **Deliverables:**

Live Q-commerce platform optimized for quick transactions

Real-time performance monitoring and error tracking setup

## **Phase 7: Post-Launch**

**Objective:** Continuously monitor and improve the platform to enhance user experience and performance.

### **Tasks:**

Monitor traffic, real-time performance metrics, and delivery success rates.

Roll out regular updates to optimize checkout, delivery, and overall performance.

Gather customer feedback and implement changes to improve user satisfaction.

### **Deliverables:**

Regular platform updates for optimization

Insights from performance metrics and customer feedback

### **Q-Commerce Focus Highlights:**

**1. Speed & Efficiency:** Instant payments, fast checkout, and real-time delivery updates are critical.

**2. Real-Time Monitoring:** Ensure immediate issue detection and performance tracking post-launch.

**3. Mobile-First:** Prioritize mobile responsiveness for quick and easy user interactions.

Post-Launch Maintenance (Q-Commerce)

### **Tasks:**

Address any user feedback and resolve bug reports with a focus on speed and usability.

Implement regular updates and new features, such as flash sales, quick promotions, or additional real-time product categories.

### **Deliverables:**

Ongoing platform maintenance optimized for fast operations.

Feature improvements and bug fixes to ensure seamless user experience.

Category-Specific Instructions (Updated for Q-Commerce)

Quick Product Features by Category

## **1. Fast Fashion**

**Real-Time Size & Color Selection:**

Enable users to instantly select available sizes and colors for clothing and accessories, displaying live stock availability.

Provide a streamlined, mobile-friendly interface to select options quickly.

**Virtual Try-On (AR/VR) for Speedy Decisions:**

Integrate Augmented Reality (AR) for instant virtual try-ons of clothes, shoes, and accessories using a mobile camera.

Offer a "mirror mode" for users to quickly visualize how products look on them before checkout.

## **2. Home Essentials**

**Quick Bundle Offers:**

- Allow users to purchase product bundles (e.g., kitchen appliances, cleaning supplies) with a focus on bulk discounts.
- Highlight real-time offers and quick-purchase options to encourage fast decision-making.

**Specifications & Quick Care Tips:**

- Provide concise specifications for items (dimensions, materials, key features) to help users make informed choices instantly.
- Include quick care tips for maintenance and product longevity.

---

## **3. Health & Wellness**

**Certifications & Lab Test Details:**

- Display certifications (e.g., FDA-approved, organic certifications) and lab test results for health products in a clear, mobile-friendly format.
- Focus on transparency to build trust and speed up purchase decisions.

### **Instant Restock Alerts:**

Notify users in real time when health and wellness items are back in stock.

---

### **Q-Commerce Key Features Focus:**

**1. Speed:** Ensure all features, including AR/VR and live stock updates, are optimized for instant use on mobile devices.

**2. Real-Time Updates:** Live updates for stock, discounts, and promotions are critical for a fast-paced Q-commerce environment.

**3. Mobile-First:** All interfaces and interactions should prioritize usability on mobile platforms.

Health & Wellness (Q-Commerce)

- Certifications & Lab Test Transparency:
- Display certifications, lab test results, and third-party testing information prominently for supplements, vitamins, and organic products.
- Ensure documents are instantly accessible on product pages to enhance trust and encourage quick purchase decisions.

### **Subscriptions for Recurring Orders:**

- Offer subscription services for recurring health and wellness products (e.g., vitamins, supplements) with automatic deliveries at regular intervals to save users time.
- Provide incentives like discounts, free shipping, or bonus items for subscription-based purchases to encourage loyalty and repeat customers.

---

## **Electronics (Q-Commerce)**

### **Quick Product Specs & Comparison Tools:**

- Showcase essential specifications (e.g., processor speed, battery life, storage capacity) in a mobile-friendly format for quick browsing.
- Integrate a side-by-side comparison tool optimized for mobile, allowing users to instantly compare products and make quick purchase decisions.
- Extended Warranty & Instant Service Plans:
- Offer extended warranties and service plans as optional add-ons at checkout, highlighting their benefits for quick decision-making.
- Provide clear details about the terms, coverage, and perks (e.g., free repairs, 24/7 priority customer service) directly on the product page to boost trust and upsell opportunities.

---

## **Collaborate and Refine :**

### **Feedback Integration:**

#### **Continuous Feedback Loop:**

- Regularly gather feedback from stakeholders (e.g., business owners, delivery partners) and end-users (e.g., customers) to identify pain points and areas for improvement.
- Use real-time feedback tools, such as mobile surveys, live chat, or post-purchase questionnaires, to understand customer expectations and satisfaction quickly.

### **Adapt and Improve:**

- Prioritize and implement critical feedback, such as faster checkout, improved delivery updates, or additional payment options, to meet user expectations.
- Monitor the impact of changes (e.g., reduced cart abandonment, faster order placement) and iterate based on performance metrics.

---

### **Q-Commerce Key Enhancements**

**1. Speed & Accessibility:** All features, from product specs to subscription management, must prioritize fast, mobile-friendly access.

**2. Transparency:** Instant access to product certifications, warranties, and real-time delivery updates fosters trust and drives quick purchase decisions.

**3. Customer-Centric Iteration:** Use real-time data and feedback to refine features quickly, ensuring the platform adapts to customer needs dynamically.

Code Reviews (Q-Commerce)

### **Peer Reviews for Speed & Scalability:**

Conduct thorough peer reviews of all code changes, focusing on high-performance, lightweight solutions tailored for real-time Q-commerce needs.

Involve team members with expertise in real-time updates, delivery APIs, and mobile optimization to ensure code quality aligns with instant service requirements.

### **Maintain Consistency for Fast Iteration:**

- Ensure the codebase adheres to standards and best practices, such as clean code, DRY principles, and modular design, to facilitate rapid feature updates.
- Use automated tools like linters to enforce consistent style guidelines, minimizing errors and saving time in fast-paced development cycles.

### **Quick Issue Identification:**

- Review code for bugs, performance bottlenecks, and compatibility issues, particularly in APIs for real-time inventory updates and delivery tracking.
- Address identified issues promptly to maintain the platform's speed and reliability before deployment.

---

### **Iterative Testing (Q-Commerce)**

#### **1:. Unit Testing**

Develop unit tests for individual components or functions to ensure they work as expected under high-demand scenarios.

Use testing frameworks like Jest for JavaScript/Next.js and Mocha for API endpoints to test core functionalities such as instant cart updates or stock availability checks.

#### **2. Integration Testing:**

Test interactions between components to verify that services like real-time delivery tracking APIs and instant checkout processes work seamlessly.

Focus on integrations critical to Q-commerce, such as the connection between the Sanity CMS API and mobile frontends.

#### **3. UI Testing:**

Use tools like Cypress or Selenium to simulate real-world scenarios, ensuring the platform is responsive and performs well across devices.

#### **Include tests for:**

- Responsiveness (mobile-first design for quick user actions).
- Interaction Elements (instant add-to-cart buttons, quick-checkout forms).

- Cross-Browser Compatibility for consistent user experience.

#### 4. Automated Testing Pipeline:

Integrate automated testing into the Continuous Integration (CI) pipeline for real-time validation of code changes before deployment.

- Automate critical flows such as:
- Real-time stock updates.
- Quick checkout processes.
- Delivery scheduling and notifications.

---

#### Key Q-Commerce Enhancements:

**1. Speed:** Code reviews and tests must prioritize lightweight and fast-loading solutions for mobile-first users.

**2. Real-Time Scalability:** Testing must account for high transaction volumes and live inventory updates.

**3. Reliability:** Automated pipelines should ensure each release maintains platform stability and performance under peak demand.

System Architecture & Documentation (Q-Commerce)

#### Regularly Update System Architecture:

Regularly update system architecture diagrams, technical workflows, and data flow documentation to reflect dynamic changes in the platform's features and infrastructure, such as real-time updates and fast order processing.



Track significant architectural changes (e.g., migration to a faster CMS, integration with additional third-party real-time APIs for delivery or payment processing) to ensure clarity and alignment among the team.

### **Agile API Documentation:**

Ensure API documentation remains current, reflecting new endpoints, request parameters, and response examples for features like real-time cart updates or fast payment processing.

Use tools like Swagger to create interactive, easy-to-navigate API documentation that developers can quickly explore, enabling fast development cycles in a Q-commerce environment.

### **User & Developer Guides for Quick Scaling:**

Maintain detailed user guides for customers and admin users, ensuring clear instructions on using Q-commerce-specific features such as instant order tracking or fast delivery scheduling.

Ensure developer documentation includes quick setup instructions, dependency management, and contribution guidelines, with a focus on scaling to handle real-time traffic spikes during peak hours.

### **Key Q-Commerce Considerations:**

**1. Real-Time Focus:** Documentation should prioritize speed, ensuring updates are made rapidly to accommodate real-time features like delivery tracking and instant order updates.

**2. Scalability:** Clear tracking of changes to the architecture ensures that Q-commerce platforms can scale efficiently, especially when adding new third-party services for faster deliveries or payments.

**3. Developer Agility:** Interactive API documentation and quick setup guides enable fast iteration, necessary for the rapid pace of Q-commerce development.

**Thank you!**

**Presentation: By Nazia Siraj**

**GICIC: 00443199**

**Friday 9:00 AM-12:00 PM**