Hackathon 2025

Day 2 : Planning the Technical Foundation

- **Transitioning to Technical Planning :**

Technical Requirements:

- Frontend Requirements

1. User-Friendly Interface:

Simple and intuitive design for seamless browsing of products.

Easy navigation with clear categories, filters, and search functionality.

2. Responsive Design:

Ensure compatibility with mobile, tablet, and desktop devices.

Optimize for fast loading times across all platforms.

Backend Requirements Using Sanity CMS

1. Sanity as the Core Backend:

Use Sanity CMS to manage all backend data for products, customers, and orders.

Leverage Sanity's real-time updates for seamless synchronization between the backend and frontend.

2. Schema Design:

Create schemas for:

Products: Attributes like name, price, category, stock availability, and images.

Customers: Name, contact details, address, and order history.

Orders: Order ID, product details, customer info,

shipping status, and payment confirmation.

Third-Party API Integrations

1. Shipment Tracking API:

Integrate with APIs like Shippo or AfterShip to provide real-time tracking updates for customers.

Fetch delivery status to display updates on the "Order Confirmation" or tracking pages.

2. Payment Gateway API:

Integrate with providers like Stripe, PayPal, or Razorpay.

Ensure secure transactions with support for multiple payment options (credit cards, wallets, UPI, etc.).

3. Additional Backend Services:

3. Additional Backend Services:

Email Notifications: Use an API like SendGrid to send

Email Notifications: Use an API like SendGrid to send confirmation emails for orders and account updates.

SMS Alerts: Integrate with services like Twilio for order status updates via text messages

4. Faster Delivery (e.g., within 15-30 minutes)

Technical Requirements:

Implement real-time order tracking with GPS integration for drivers and customers.

Build an efficient route optimization algorithm to minimize delivery times.

Design a distributed warehouse management system to place inventory closer to high-demand areas.

Ensure scalable infrastructure (cloud-based) to handle high traffic during peak hours without latency.

Integrate driver availability tracking to allocate resources dynamically.

Integrate driver availability tracking to allocate resources dynamically.

5. Driver and Workforce Management

Build a driver app with features like order notifications, navigation, and earnings tracking.

Implement real-time driver performance monitoring and feedback systems.

Integrate shift scheduling systems to optimize workforce availability.

6. Customer Retention and Marketing

Technical Requirements:

Develop a push notification system for personalized promotions and offers.

Integrate customer feedback and rating systems to gather insights.

Build a CRM system for managing loyalty programs and customer segmentation.

Use data analytics platforms to track customer behavior and optimize campaigns.

•System Architecture Design

Components Overview:

1. Frontend (Next.js):

Handles user interactions, renders pages, and communicates with the backend via APIs.

2. Sanity CMS:

Acts as the central content management system for products, orders, and customer data.

3. Product Data API:

Fetches product details from Sanity CMS for the frontend.

4. Third-Party APIs:

External services for shipment tracking, payment processing, and notifications.

5. Payment Gateway:

Manages secure payment transactions.

6. Shipment Tracking API:

Provides real-time order tracking information.

7. Notification Service:

Sends order updates via email or SMS

Workflow Explanation

1. Frontend to Sanity CMS:

The Next.js frontend communicates with the Sanity CMS via the Product Data API to retrieve and display product information dynamically.

2. Sanity CMS to Product Data API:

Sanity CMS acts as the database for products, customer accounts, and orders.

Product Data API enables seamless integration between the frontend and CMS

3. Frontend to Payment Gateway:

During checkout, payment details are sent securely from the frontend to the Payment Gateway API (e.g., Stripe, PayPal).

Upon successful payment, the gateway returns a transaction confirmation.

4. Frontend to Shipment Tracking API:

Once an order is placed, the frontend fetches real-tim tracking data from the shipment tracking API (e.g., AfterShip).

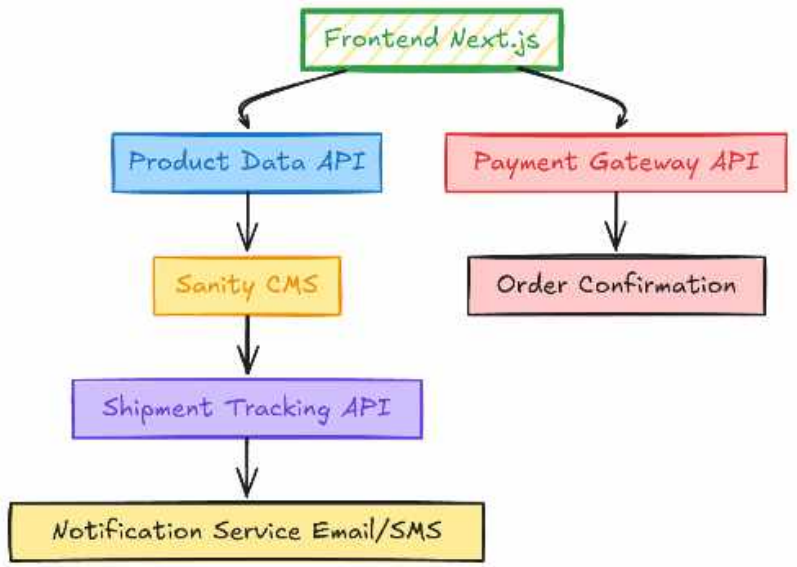
Updates are displayed on the Order Confirmation and Tracking pages.

5. Sanity CMS to Notification Service:

After an order is created, Sanity CMS triggers the Notification Service to send confirmation emails/SMS

After an order is created, Sanity CMS triggers the Notification Service to send confirmation emails/SMS to the customer.

Diagram:



•Plan API Requirements :

1. Express Delivery Status

Endpoint Name: /express-delivery-status

Method: GET

Description: Fetches the status of an express delivery based on a unique delivery ID.

Parameters:

delivery_id: (Required) The unique identifier for the delivery.

Response:

status: The current status of the delivery (e.g., "In Progress", "Delivered", "Delayed").

estimated_delivery_time: The estimated time of delivery.

current_location: The current location of the delivery.

current_location: The current location of the delivery.

2. Product Availability

Endpoint Name: /product-availability

Method: GET

Description: Returns the availability status of a produ
at the nearest warehouse.

Parameters:

product_id: (Required) The unique product identifier.

location: (Optional) User's location to prioritize neares
warehouses.

Response:

available: Boolean indicating availability

quantity: Available quantity in stock.

warehouse_location: Location of the warehouse.

3. Order Tracking

Endpoint Name: /order-tracking

Method: GET

Description: Fetches real-time tracking data for a customer's order.

Parameters:

order_id: (Required) The unique identifier for the order.

Response:

status: Current order status (e.g., "Processing", "Shipped", "Out for Delivery").

"Shipped", "Out for Delivery").

shipment: Tracking number and shipping provider.

expected_delivery: The expected delivery date.

4. Customer Support Request

Endpoint Name: /support-request

Method: POST

Description: Allows customers to submit support requests.

Body:

customer_id: (Required) Customer identifier.

issue_type: (Required) Type of issue (e.g., "Delivery", "Payment", "Product Issue").

description: (Required) Detailed description of the issue.

Response:

request_id: Unique identifier for the support request.

status: Status of the request (e.g., "Received", "In Progress").

5. Place Order

Endpoint Name: /place-order

Method: POST

Description: Submits an order for express delivery.

Body:

customer_id: (Required) Customer's unique ID.

product_ids: (Required) List of product IDs to be ordered.

address: (Required) Delivery address.

payment_method: (Required) Payment details.

Response:

order_id: Unique identifier for the order.

total_price: Total order cost.

delivery_time: Estimated delivery time.

- Technical Document:

1. System Architecture Document

Overview:

The system is designed for a Q-Commerce platform, where users can place express delivery orders and track them in real-time. It involves various microservices that interact with each other through APIs to handle user requests, orders, payments, delivery statuses, and support tickets.

The architecture follows a modular, service-oriented approach to ensure scalability, reliability, and ease of maintenance.

Components:

1. Frontend:

User-facing application (Web and Mobile)

Components: Order Placement, Product Search, Order Tracking, Support Request.

2. API Gateway:

Central entry point for all client requests.

Routes the requests to the appropriate microservices.

3. Microservices:

User Service: Manages user data, authentication, and

and profiles.

Order Service: Handles order creation, updates, and status tracking.

Delivery Service: Manages real-time delivery status an delivery logistics.

Inventory Service: Manages product availability and stock data.

Payment Service: Processes payments and manages transactions.

Support Service: Manages support requests and customer service tickets.

4. Database:

Relational Database (e.g., PostgreSQL) for structured data.

Key-value store (e.g., Redis) for session and cache management.

Key-value store (e.g., Redis) for session and cache management.

5. External Integrations:

Payment Gateway (e.g., Stripe or PayPal)

Shipping Carrier API (for real-time delivery tracking)

6. Security:

OAuth 2.0 for user authentication and authorization.

SSL/TLS encryption for data transmission.

Interaction Flow:

1. A user interacts with the frontend, initiating an action (e.g., placing an order).

an action (e.g., placing an order).

2. The frontend makes an API request to the API Gateway.

3. The API Gateway routes the request to the relevant microservice (e.g., Order Service).

4. The microservice processes the request (e.g., creating an order), interacts with databases as needed and returns the response.

5. The frontend receives the response and updates the user interface accordingly.

2. API Specification Document

1. Express Delivery Status

Endpoint Name: /express-delivery-status

Method: GET

Description: Retrieves the status of an express delivery based on the delivery ID.

Request Parameters:

delivery_id: (string) The unique identifier for the delivery.

Response:


```
{
  "status": "In Progress",
  "estimated_delivery_time": "2025-01-17T15:30:00Z",
  "current_location": "Warehouse 3"
}
```


2. Place Order

Endpoint Name: /place-order

Method: POST

Method: POST

Description: Submits an order for express delivery.

Request Body:

```
{
  "customer_id": "12345",
  "product_ids": ["prod_001", "prod_002"],
  "address": "123 Main Street, City, Country",
  "payment_method": "credit_card"
}
```

Response:

```
{
  "order_id": "ORD123456",
  "total_price": 49.99,
  "delivery_time": "2025-01-17T15:30:00Z"
}
```

3. Order Tracking

Endpoint Name: /order-tracking

Endpoint Name: /order-tracking

Method: GET

Description: Fetches real-time tracking data for a customer's order.

Request Parameters:

order_id: (string) The unique identifier for the order.

Response:


```
{
  "status": "Shipped",
  "shipment": "Tracking12345",
  "expected_delivery": "2025-01-17T15:30:00Z"
}
```


4. Customer Support Request

Endpoint Name: /support-request

Endpoint Name: /support-request

Method: POST

Description: Submits a support request for customer assistance.

Request Body:

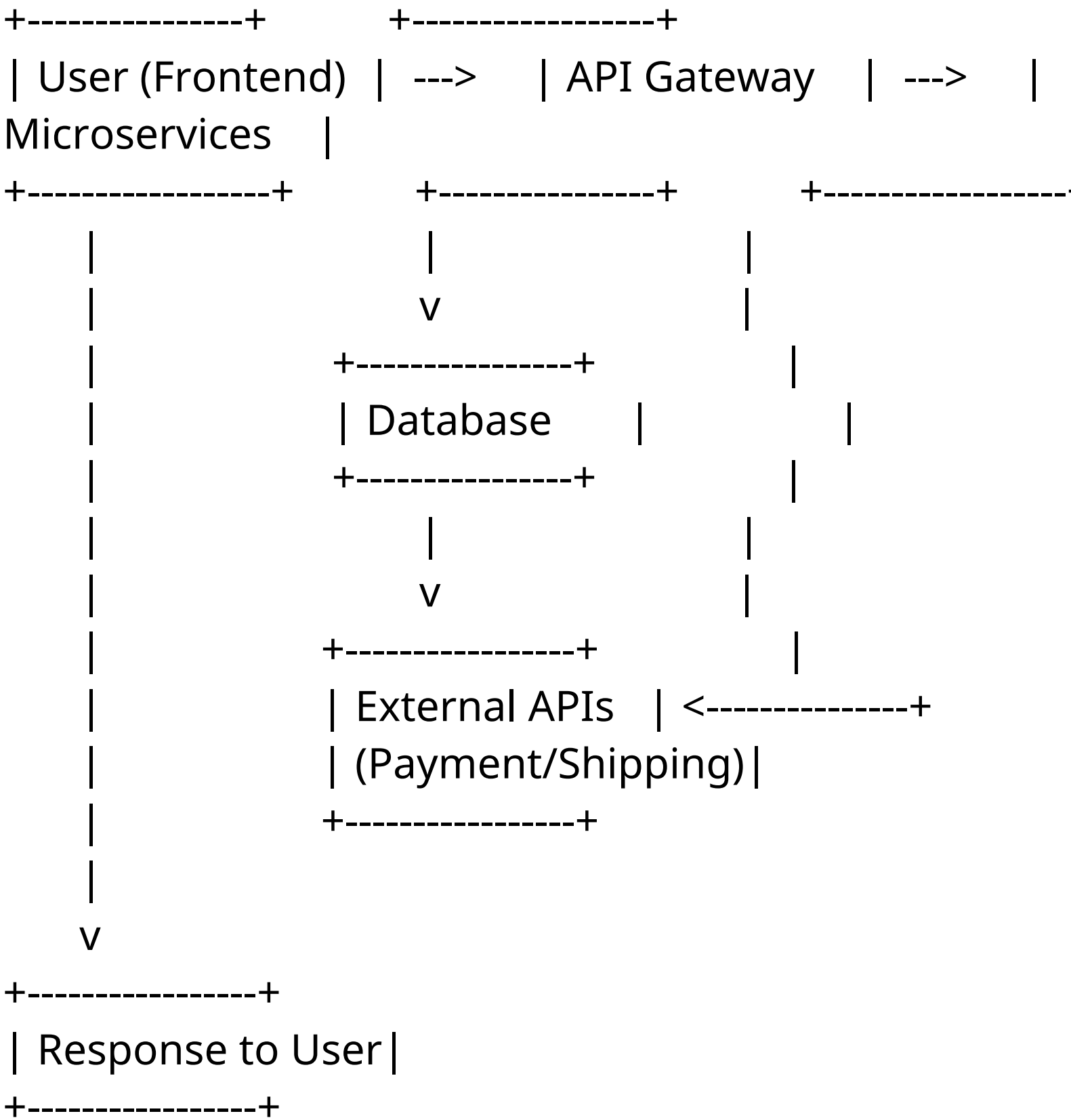
```
{
  "customer_id": "12345",
  "issue_type": "Delivery",
  "description": "The delivery was delayed."
}
```

Response:

```
{
  "request_id": "REQ12345",
  "status": "Received"
}
```

3. Workflow Diagram

Below is a simplified visualization of the user interaction and data flow through the system



4. Data Schema Design

Entities:

User:

user_id: Unique identifier (Primary Key)

name: Customer’s name

email: Customer’s email address

password_hash: Encrypted password

address: Delivery address

Product:

product_id: Unique identifier (Primary Key)

name: Product name

price: Price of the product

stock_quantity: Quantity available in stock

stock_quantity: Quantity available in stock

Order:

order_id: Unique identifier (Primary Key)

customer_id: Foreign key referencing User(user_id)

product_ids: List of products included in the order

total_price: Total order cost

order_status: Current order status (e.g., "Processing", "Shipped")

delivery_time: Estimated delivery time

Payment:

payment_id: Unique identifier (Primary Key)

order_id: Foreign key referencing Order(order_id)

payment_method: Method used for payment (e.g., "credit_card")

payment_status: Status of the payment (e.g., "Completed", "Pending")

Support Request:

request_id: Unique identifier (Primary Key)

customer_id: Foreign key referencing User(user_id)

issue_type: Type of the issue (e.g., "Delivery", "Payment")

description: Issue description

status: Status of the request (e.g., "Received", "In Progress")

5. Technical Roadmap

Phase 1: Planning & Requirements

Define system architecture and key services.

Identify all required APIs and data models.

Research third-party services (e.g., Payment Gateway, Shipping API).

Phase 2: Development & Integration

Develop core services (User, Order, Inventory, Payment).

Integrate third-party APIs (Payment Gateway, Shipping Carrier).

Implement the API Gateway for routing requests.

Phase 3: Testing

Conduct unit tests for individual services.

Conduct unit tests for individual services.

Perform integration testing for API interactions.

Test security measures (e.g., OAuth, SSL).

Phase 4: Deployment

Deploy the system on staging environment for final testing.

Launch production environment.

Phase 5: Post-launch Monitoring & Maintenance

Set up logging and monitoring.

Continuously improve based on user feedback.

Milestones:

M1: Complete system design and architecture.

M2: Develop and test core microservices.

M3: Integrate third-party APIs.

M4: Finalize testing and launch.