

# Order Tracking & Notification System

## Problems

- Customers cannot access real-time information about their order status.
- There is a lack of integration with shipping companies.
- Delivery times are unclear or unpredictable.
- Some customers want to receive order status updates via SMS or email.

---

## Business Needs and Success Criteria

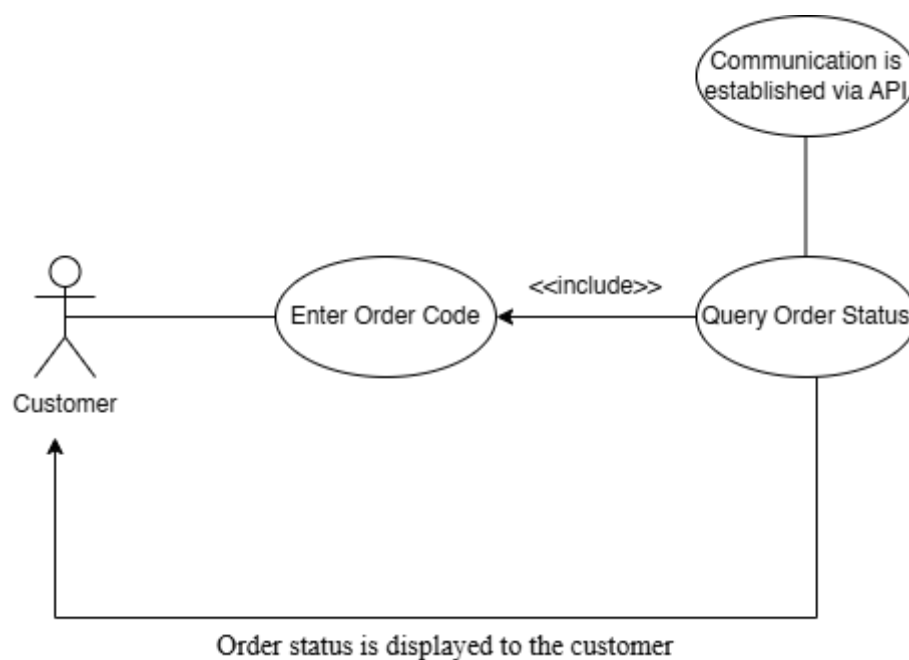
- Order status should be viewable in real time (The customer panel must display live shipment status).
- Integration with shipping companies is required (The system should automatically retrieve shipment updates via API).
- Estimated delivery time should be provided (The system should display an estimated delivery date when the order is placed).
- Notification preferences should be offered (Customers should receive updates via SMS or email, based on their selection).

ID	Use Case Name	Priority: (Lowest: 1, Highest: 5)	Explanation
UC1	Display Order Status	5	Viewing the current status of the order live on the customer panel.
UC2	Shipping Company Integration	5	Shipment tracking via API.
UC3	SMS / Email Notifications	5	Automated notification based on order status.
UC4	Estimated Delivery Time	4	Displaying estimated delivery time to the user.
UC5	Test and QA Process	5	Testing of all system functions.

Use Case Priority List

The **Use Case Priority List** table presented above allows for a structured prioritization of the use cases included within the scope of the project, based on specific criteria. Each row represents a use case, with details such as descriptions, priority levels, and other relevant information provided.

The table clearly indicates which functionalities are more critical in terms of user experience and business objectives, which ones require more complex technical infrastructure, and how much time they may take to implement. This structure enables the project team to allocate resources more efficiently and determine the optimal sequence of development steps. Additionally, it contributes to transparent communication with stakeholders and supports better management of project expectations.



**Use Case Diagram for Viewing Order Status**

This diagram is a Use Case Diagram that illustrates the process by which a customer retrieves the status of their order. The actor in the diagram is the "Customer," and the interaction with the system begins with the use case "Enter Order Code." Once the customer inputs the order code into the system, the system proceeds to the "Query Order Status" use case using that code. This use case communicates with an external system—specifically, the shipping company's service—via an API.

The "Query Order Status" function allows the system to retrieve the **"Current Status of the Order."** After this information is obtained, the current status of the shipment is displayed to the customer. In the diagram, the <<include>> relationship is used to indicate that the "Enter Order Code" use case necessarily includes the "Query Order Status" functionality. This structure effectively reflects the relationships between use cases and the underlying data flow within the system in a clear and concise manner.

## **PROBLEM:** Lack of Real-Time Order Tracking, Delivery Transparency, and Shipping Integration

---

### **What Does This Mean?**

Customers are unable to view the real-time status of their orders after making a purchase. They don't know whether the order is being prepared, shipped, or delivered, nor do they have access to an estimated delivery time. Even worse, these status updates are often not reflected in the interface, arrive too late, or are not sent at all.

Additionally, some customers wish to receive status notifications via SMS or email, but the system does not provide automated updates. At the same time, the platform lacks proper integration with shipping companies, which prevents automatic data exchange, resulting in manual and inefficient tracking.

---

### **Root Technical Causes**

<b>Cause</b>	<b>Explanation</b>
Missing or poor user interface	No visual timeline or tracking component for order status.
Delayed or missing data updates	Shipment status is not updated promptly or consistently in the system.
No API integration with shipping	Shipping APIs are not connected to the system for tracking updates.
No webhook infrastructure	Shipping providers cannot push real-time updates to the system.
No estimated delivery mechanism	System doesn't calculate or show an expected delivery time.
Orders not linked to tracking	Order objects lack references to tracking numbers or shipping records.
Incompatible data formats	Shipping data is not properly parsed or converted (e.g., XML → JSON).
No scheduler or background jobs	System doesn't fetch updates periodically from shipping APIs.
No notification system	Status changes are not triggering SMS or email notifications.
Missing security configuration	API keys, tokens, or whitelisting are not implemented properly.

---

## Business and User Impact

Problem	Result
Customers are unaware of order progress	Frustration, repeated support tickets, and poor experience.
No visibility into delivery timeframe	Decreased trust and perceived unreliability.
Call center overload	Support staff wastes time answering basic “Where is my order?” queries.
Missed notifications cause confusion	Customers miss delivery attempts or stay uninformed.
Lower satisfaction and retention	Reduced loyalty and likelihood of repeat purchases.

---

## Recommended Solutions

### User Experience Enhancements:

- Add a shipment timeline UI showing:  
Order Received → In Preparation → Shipped → Delivered
- Display estimated delivery date at the time of purchase and in the order details.
- Allow customers to select notification preferences (SMS, email, push).
- Trigger automated status updates when milestones are reached.

### Backend & System Enhancements:

- Add status and estimated\_delivery fields to order data models.
  - Integrate with shipping APIs to send order data and receive tracking info.
  - Set up webhook endpoints to receive real-time status updates.
  - Create scheduled jobs to fetch status updates regularly via API.
  - Design a notification service to send alerts based on status changes.
  - Store a log of shipment status changes for historical visibility.
  - Implement failover/retry logic in case of API/webhook failures.
  - Ensure secure communication via API keys, tokens, or IP restrictions.
  - Map and transform external data formats (e.g., XML to JSON).
-

## Conclusion

By addressing the combined challenges of limited order visibility, absent delivery time estimates, and lack of shipping integration, this solution ensures a complete and transparent shipment experience. Automated notifications, real-time tracking, and a clear delivery timeline will not only reduce operational overhead but also build trust and satisfaction across the customer journey—making the system scalable, modern, and customer-centric.

## USER STORY

➔ As a customer, I want to track my shipment, so that I can know when it will arrive.

## ACCEPTANCE CRITERIA

- The customer must see the current status of the shipment.
- The system must update the status every 30 minutes.
- A notification must be sent when the status is “Delivered”.

## SPRINT PLAN FOR CARGO INTEGRATION

### Sprint 1: API Integration & Backend Setup

Objective:

To establish a working integration between the e-commerce system and the shipping provider by connecting to their API and setting up a webhook to receive status updates.

Planned Tasks:

1. Review the shipping company’s API documentation
2. Develop API connection to retrieve tracking data
3. Create a webhook endpoint to receive delivery updates
4. Test the API integration with a sample shipment
5. Handle error cases and timeouts (basic retry logic)

Expected Duration:

5 working days

Sprint Goal:

System is technically integrated with the carrier and can retrieve tracking status via API/webhook.

---

## **Sprint 2: Shipment Status Tracking & Notifications**

### **Objective:**

To enable customers to track their orders in real time and receive notifications when the shipment status changes.

### **Planned Tasks:**

1. Extend the order and shipment database model to include tracking\_status, carrier\_id, tracking\_number
2. Display current shipment status on the customer panel
3. Trigger SMS and email notifications upon delivery updates
4. Perform user acceptance testing (UAT)
5. Finalize exception handling for failed API/webhook events

### **Expected Duration:**

5 working days

### **Sprint Goal:**

Customers can see up-to-date shipment information and are notified automatically when the order is delivered.

---

## **Sprint 3: Reporting & System Improvements**

### **Objective:**

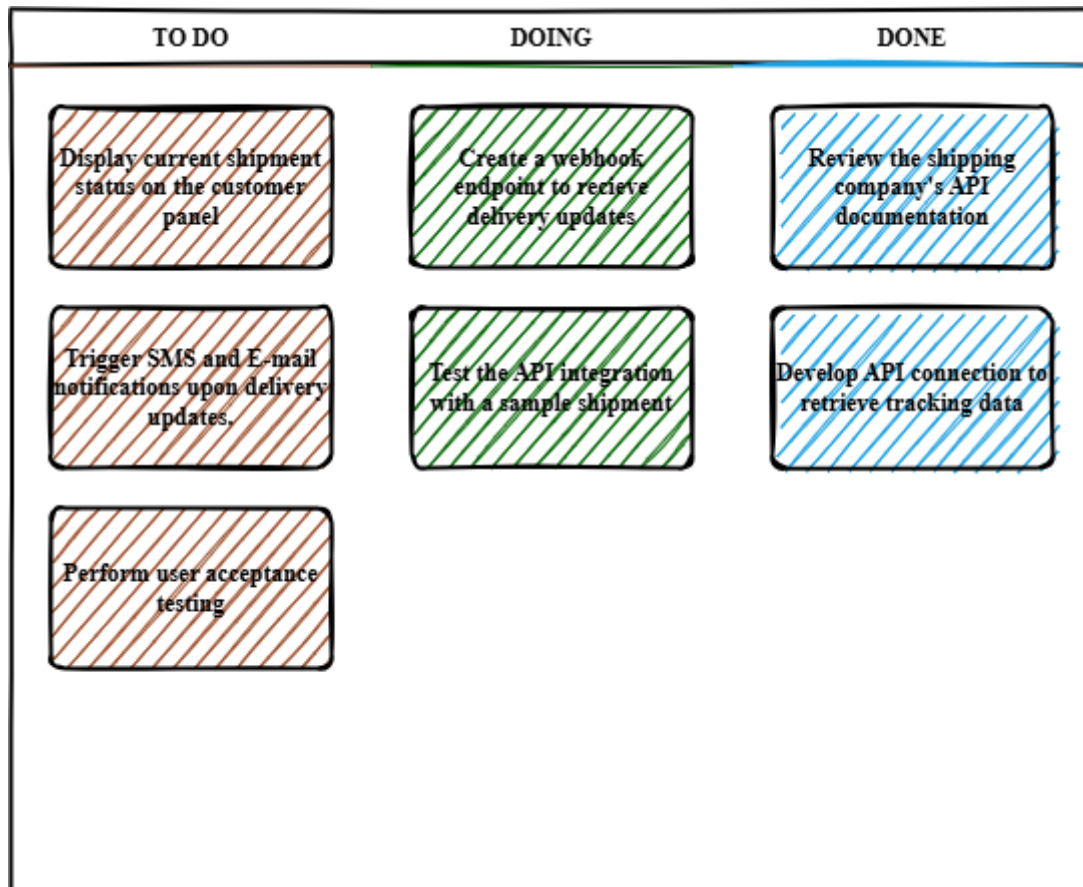
To enhance reliability and operational efficiency through logging, reporting, and failover mechanisms.

### **Planned Tasks:**

1. Log all status updates for audit purposes
2. Implement delivery performance reports (e.g., average delivery time)
3. Improve retry logic and error handling for API failures
4. Optimize database indexing for faster status lookup

### **Expected Duration:**

3–4 working days



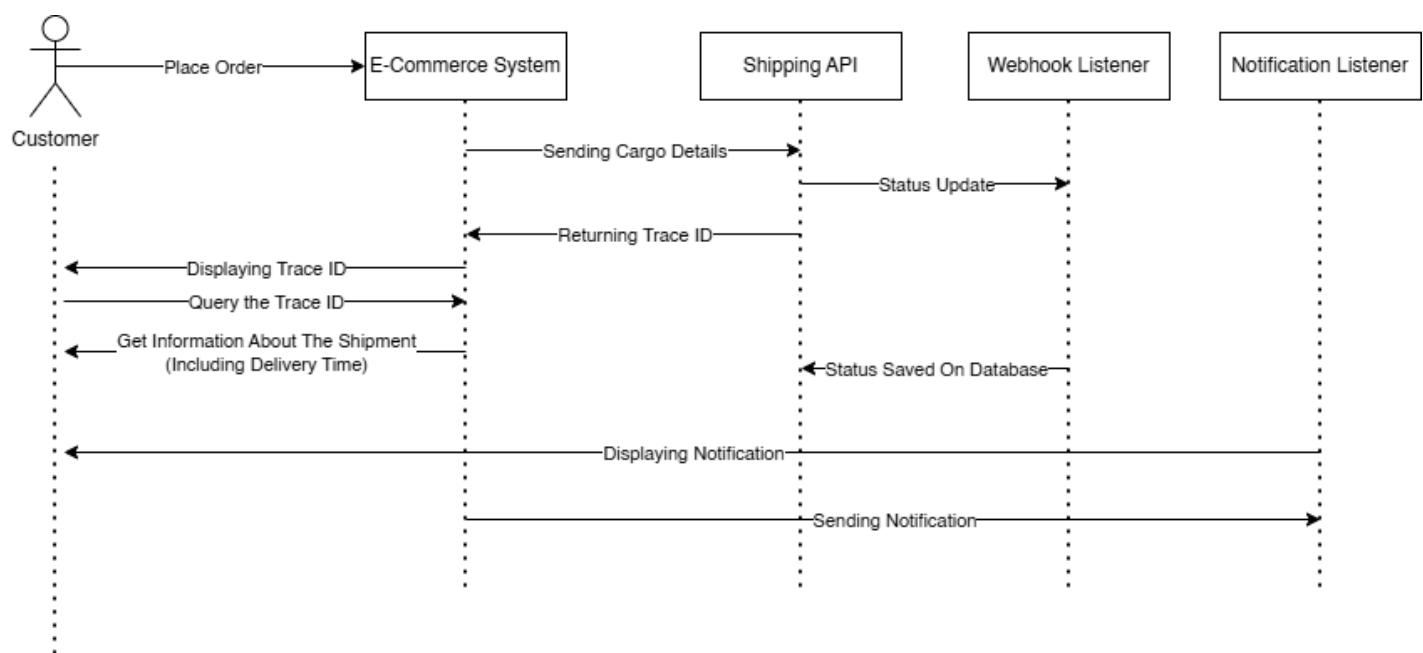
**Sample Kanban Board of the sprint plan.**

Test Case ID	Test Description	Input/Action	Expected Result
TC001	Order tracking is visible to the customer.	Customer places an order and visits "My Orders" page.	Tracking number and current shipment status are displayed.
TC002	Shipment status is updated from the carrier	Carrier sends webhook with status = "Out for delivery"	Status is updated in the database and shown on customer interface.
TC003	Delivery notification is triggered.	Shipment status changes to "Delivered"	Customer receives SMS and email notification.
TC004	System retries API call on failure.	Carrier API is temporarily unavailable.	System retries the request up to 3 times before logging the failure.
TC005	Carrier is correctly linked to the shipment.	Shipment is created with a selected carrier.	Carrier name and logo appear in the order details.

**Table Of Test Scenario**

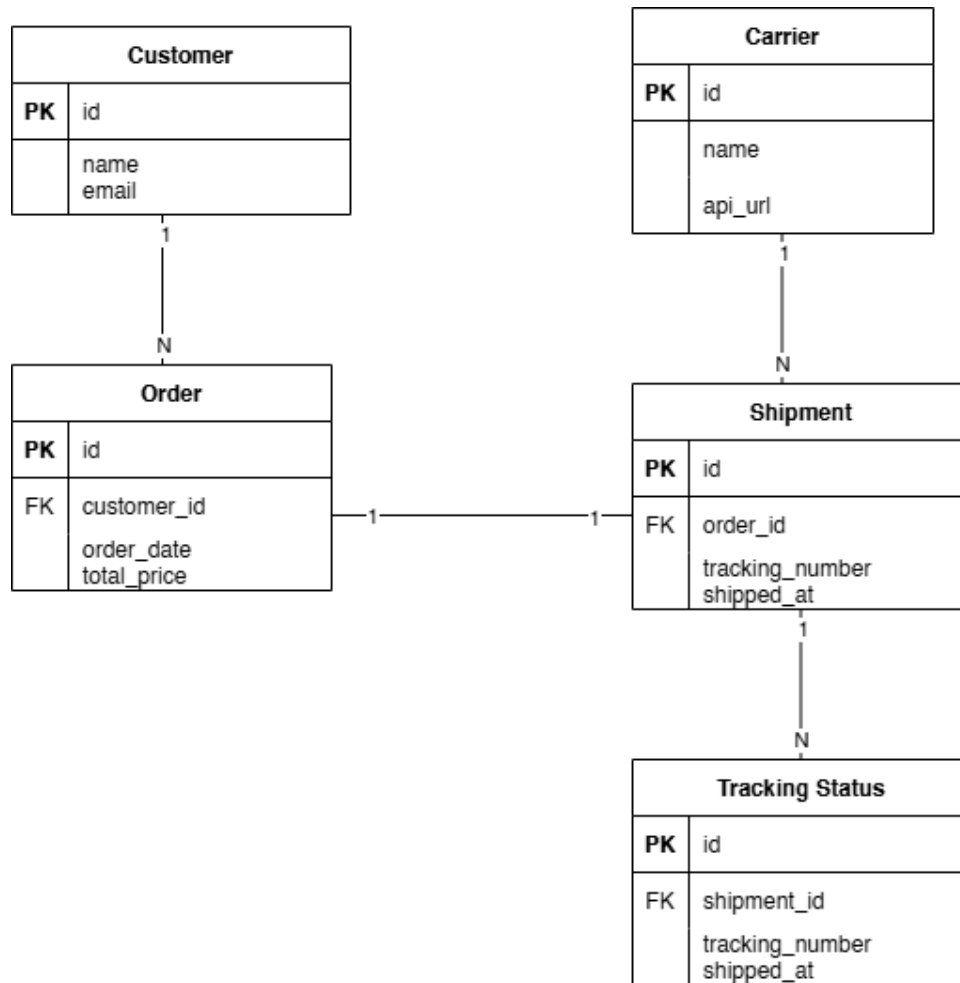
The test scenario table outlines critical user acceptance test cases designed to verify the functionality of the cargo integration system. Each scenario defines a specific user action or system event, along with the expected outcome to ensure that key features—such as real-time shipment tracking, status updates via webhook, and delivery notifications—work correctly. These test cases help validate both the technical reliability of the integration and the end-user experience. By running through these scenarios, we can confirm that the system performs as intended under real-world conditions, minimizing potential issues before deployment.

Let’s inspect this “Cargo Integration” situation with diagrams:



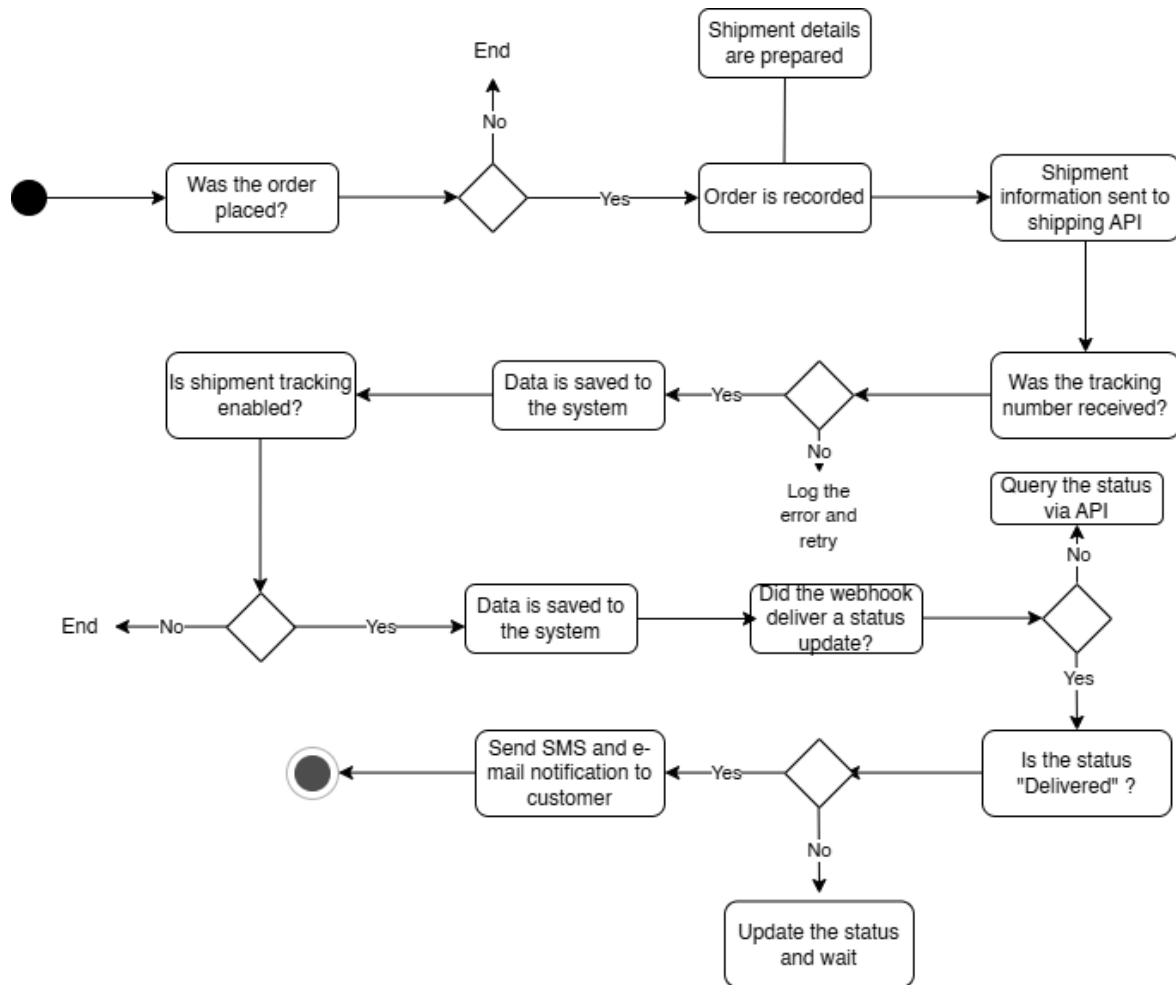
**Sequence Diagram Of Cargo Integration**

The sequence diagram illustrates the ideal and most effective scenario for managing shipment tracking and customer notifications in an e-commerce system. It outlines the complete interaction flow between the customer, e-commerce platform, shipping API, webhook listener, and notification service. The process begins when a customer places an order, after which the e-commerce system communicates with the shipping API to initiate the shipment and retrieve a unique tracking number (trace ID). This trace ID is presented to the customer for reference and manual tracking if needed. As the shipment progresses, the shipping API sends real-time status updates to the system via a webhook, which ensures timely data synchronization. The webhook listener receives and stores the updated status, including estimated delivery time, into the system database. Based on these updates, the notification listener triggers a message—via SMS or email—to inform the customer of the current shipment status. This architecture ensures that the customer can access consistent information across all channels, whether through direct system interaction or proactive notifications. By combining API communication, event-driven webhooks, and modular notification handling, this sequence represents a best-practice implementation for enhancing both operational efficiency and customer satisfaction in shipment tracking systems.



**E-R Relationship Diagram of Cargo Integration System**

The Entity-Relationship (ER) diagram illustrates the core structure of a cargo integration system by outlining key entities and their relationships. A Customer can place multiple Orders, establishing a one-to-many (1:N) relationship between Customer and Order. Each Order is linked to a single Shipment, representing a one-to-one (1:1) relationship. The Shipment is handled by a Carrier, where multiple shipments can be associated with one carrier—this forms a many-to-one (N:1) relationship. Additionally, a Shipment can have multiple TrackingStatus updates over time, defining another one-to-many (1:N) relationship. While there is no direct relationship between Customer and Carrier, their connection is established indirectly through the Order and Shipment entities. This model enables structured tracking of customer orders, shipping processes, and carrier activity within the system.



### Activity Diagram of Cargo Integration System

The activity diagram illustrates the ideal workflow of an e-commerce order fulfillment and shipment tracking process. It begins with the placement of an order by the customer. If the order is confirmed, the system prepares the shipment details and records the order, followed by sending the shipment data to the shipping API. Once a tracking number is received, the system saves it and checks whether shipment tracking is enabled. If enabled, the system proceeds to monitor status updates either through a webhook or, in the absence of one, by querying the shipping API. Upon receiving a status update, the system verifies whether the shipment status is “Delivered.” If not, it continues to monitor and update the status. When the status is confirmed as delivered, an SMS and email notification is sent to the customer. The diagram uses clear decision nodes (diamonds) to depict all conditional steps, ensuring transparency in how the process handles failures, retries, and updates. This structure supports a reliable, event-driven shipment tracking system that enhances user communication and delivery transparency.

## **Additional Feature Suggestions**

---

### **1. AI-Based Delivery Time Estimation**

Instead of relying solely on carrier-provided estimates, the system can analyze past deliveries and provide dynamic delivery time predictions.

Benefit: Increases accuracy of estimated delivery dates and introduces machine learning into the project.

---

### **2. Shipment Delay Detection & Alerting**

The system can detect when an order remains too long in the same shipment status (e.g., more than 72 hours) and automatically notify the admin or support team.

Benefit: Helps proactively address issues before customers complain.

---

### **3. Live Map-Based Shipment Tracking**

By integrating with mapping services (e.g., Google Maps), the system can show the real-time location of the shipment based on carrier data.

Benefit: Adds visual value and enhances user transparency.

---

### **4. Post-Delivery Feedback Collection**

After delivery is complete, the system prompts the customer to rate the delivery experience and optionally leave a comment.

Benefit: Enables quality control and customer satisfaction analysis.

---

### **5. Secure Tracking Link with Expiration**

The system can generate time-limited, token-based tracking URLs for added security, especially useful for corporate or sensitive deliveries.

Benefit: Improves data privacy and access control.

---

### **6. Admin Dashboard – Performance Reports**

A separate interface for administrators showing metrics like:

- Average delivery time
- Number of delayed orders
- Carrier performance comparison

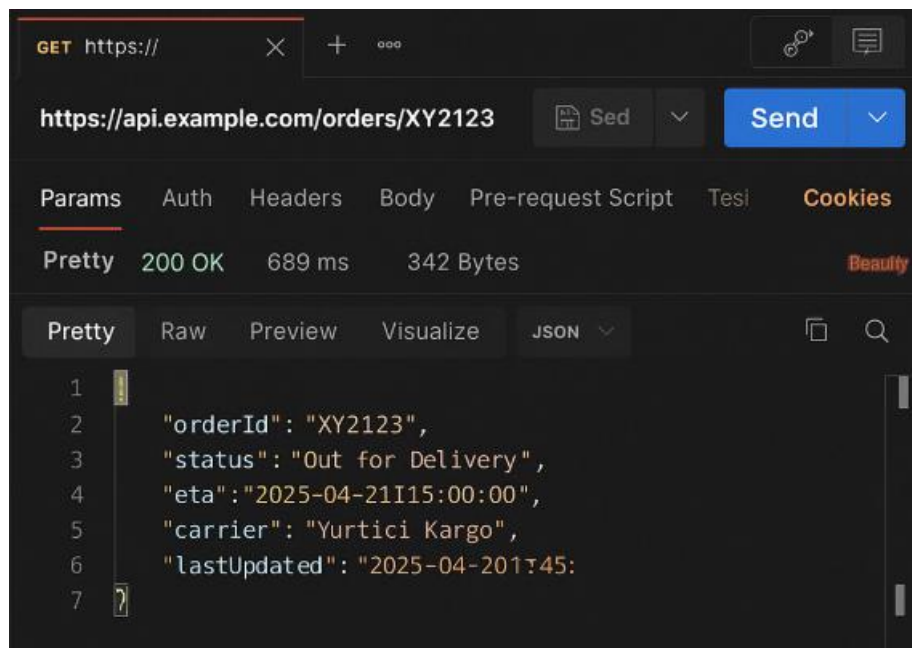
Benefit: Supports decision-making with operational data.

## Product Backlog

ID	User Story	Priority	Status	Story Points	Note
PB-01	As a user, I want to see my order status in real time.	High	To do	5	Timeline UI
PB-02	As a system, I need to retrieve status from the shipping API.	High	To do	8	REST Entegration
PB-03	As a user, I want to receive SMS/email when my order status changes.	Medium	To do	5	Notification System must be built
PB-04	As a user, I want to see the estimated delivery time.	Medium	To do	3	ETA must be taken from API
PB-05	As a PO, I want order status changes to be logged for future traceability.	Low	To do	3	Logging must be done

The product backlog acts as a living to-do list for the development team, outlining all the features, improvements, and technical needs required to complete the system. Items in the backlog are written from the user's point of view and help the team understand what matters most. Each task is given a priority and a rough estimate of effort, allowing the team to focus on the most important work first. By organizing the work this way, the project stays aligned with both the user's expectations and the overall goals of the product. In the below Product Backlog Table, we can observe the extra feautres that can be added to the plan.

ID	User Story	Priority	Status	Story Points	Note
PB-06	As a user, I want the system to provide a more accurate delivery estimate based on past deliveries.	Medium	To do	8	Prediction with AI/ML
PB-07	As a system, I want to notify the admin if a shipment is stuck too long in the same status	High	To do	5	Products that remains same status will be logged.



### Postman-style API mock-up

This Postman-style API mock-up illustrates how the Order Tracking & Notification System interacts with external shipping services through RESTful APIs. The mock-up demonstrates a typical GET request made by the system to retrieve the status of a specific order using its tracking number. The corresponding JSON response includes key details such as the order status, estimated delivery time (ETA), and carrier information. Additionally, the mock-up showcases how webhook events are received from the shipping provider when a status update occurs, ensuring real-time synchronization between the shipment process and the system's internal records. This simulation effectively represents the data exchange mechanism without requiring a live backend, helping visualize how API communication enables seamless order tracking functionality.

Order Number

**Shipment Status**

Shipped      Out for delivery      Delivered

**Estimated Delivery**

Thursday, May 2

**Notification Preferences**

☐ SMS

☐ Email

**UI mock-up**

The UI mock-up presented here demonstrates the visual flow and user experience of the Order Tracking & Notification System. It provides a simulated view of how customers interact with the platform, starting from entering their order number to viewing real-time shipment status and estimated delivery details. The design includes key interface elements such as input fields, status timelines, and notification preferences, offering a clear and intuitive user journey. This mock-up helps visualize the final product's appearance without requiring a working application, ensuring that both technical teams and stakeholders share a common understanding of the system's design and functionality.

## **Abstract**

This project focuses on building a responsive and efficient system that allows customers to track their orders in real time after completing a purchase on an e-commerce platform. In the current process, customers often rely on customer service to check the status of their orders, which not only delays information flow but also increases the workload on support teams.

Additionally, the lack of connection between the platform and shipping providers, along with the absence of delivery estimates, contributes to a poor post-purchase experience.

To address these issues, the system is designed to automate shipment tracking by connecting to external carrier APIs and gathering real-time delivery updates using webhooks. It includes key features such as live order status tracking, estimated delivery predictions, and timely notifications sent via SMS or email. Each order moves through a well-defined lifecycle, with all status changes recorded to maintain transparency and ensure traceability.

The project was approached using Agile methodology, focusing on user-centered development through prioritized product backlog items, structured sprint planning, and clear functional requirements. Supporting models such as activity flows, use case diagrams, and state transitions were also developed to guide the implementation process.

In summary, this solution not only enhances customer satisfaction by providing visibility and timely updates, but also reduces support center dependency, making the overall e-commerce delivery process more scalable, automated, and reliable.