Project Management Assessment

Candidate: Pragati Regmi
Position: Project Manager
Date: 28th March, 2025

TASK 1: Project Knowledge

a. Describe the immediate steps you would take to address the data inconsistencies in the ERP integration.

To address the data inconsistencies in the ERP integration, I would prefer **RAID log method**

(Risks, Assumptions, Issues, Dependencies) to systematically track and resolve the problem.

No	Category	Description	Priority	Next Actions
1	Risk	Mismatch in the data between ERP (Microsoft NAV) and the portal which might lead to wrong product availability and the error in the order	Critical	1). Audit all the data 2). Add validation checks before syncing.
2	Risk	Two-way sync between ERP and portal is complex which might lead to project delay.	High	1). Split the sync process into smaller steps. 2). Use a one-way sync as a temporary fix and add more developers if required
3	Assumption	Distributor will use the portal even if some of the features are not available.	Medium	1). Keep distributors updated about new features.
4	Assumption	The ERP team will provide API docs and support on time.	Medium	1). Retrospective with the ERP team and check

				for missing API functions early and request the fixes.
5	Issue	Distributors want extra features like bulk order uploads and detailed product specification	High	1). Use the MoSCoW method to decide which features to build first.
6	Issues	The portal is slow when handling large orders during testing.	Critical	1). Improve the database queries and API calls and run more tests to fix performance issues.
7	Dependency	Legal team must approve distributor agreements before onboarding.	Critical	1). Work with the legal team to finalize documents

Furthermore, to address the data inconsistencies in ERP integration, we require involving the right people and defining their roles clearly. So, below is the breakdown of the role:

1. ERP Team

   Role: Understand Microsoft NAV's data structure and provide support.

   Responsibilities:

- Explain how data is stored in Microsoft NAV (e.g., product codes, pricing, customer details).
- Provide clean and accurate data for integration (e.g., no duplicates or missing records).
- Help the development team understand how to extract and use data from the ERP system.
- Fix any data-related issues in the ERP system (e.g., correcting wrong product codes).

2. Developers

   Role: Build and fix integration between the ERP and B2B portal

Responsibilities:

- Write code to sync data between Microsoft NAV and the B2B Portal.
- Fix data mapping issues (e.g., ensure product codes in ERP match those in the portal).
- Add validation checks to prevent errors (e.g., reject orders with missing customer details).
- Create tools to detect and fix data inconsistencies automatically.

3. QA Engineers

   Role: Test the integration to ensure the system work as expected.

   Responsibilities:

- Test if data syncs correctly between the ERP and the portal (e.g., product details, pricing).
- Check for errors like missing data, duplicates, or mismatched fields.
- Test how the system handles large orders or unusual data (e.g., special characters).
- Verify that fixes for data inconsistencies actually work.

4. Project Manager

   Role: Keep the project on track and ensures everyone work together

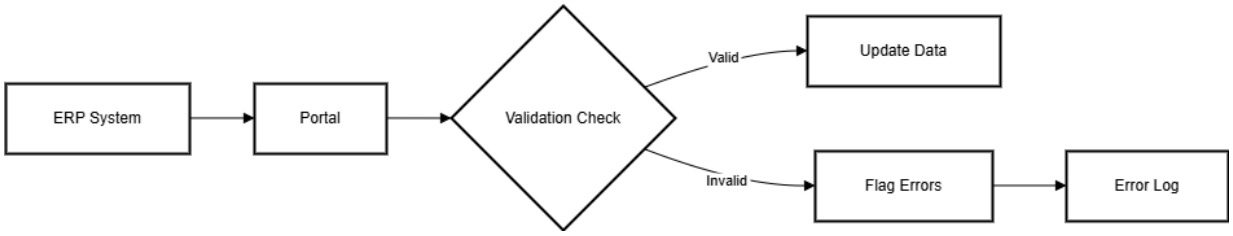
   Responsibilities

- Create a timeline for the integration and track progress.
- Coordinate between the ERP team, developers, and QA engineers.
- Use tools like the RAID Log to track risks, issues, and dependencies.
- Escalate problems to higher management if there are delays or roadblocks.
- Communicate updates to stakeholders (e.g., business teams, distributors).

5. End-Users

- Role: Use the B2B Portal during testing to check if data is accurate (e.g., product details, pricing).
- Report any issues (e.g., wrong product codes, missing orders).
- Confirm that the portal meets their needs.

How bad data does enters???



b. How would you assess and prioritise the additional feature requests from distributors?

To assess and prioritize feature requests, I would use the structured steps as follows:

Step 1: Gather and Document feature requests

- Use surveys, interviews, or feedback forms to gather feature requests from distributors.
- List all requested features with details (e.g., bulk order uploads, detailed product specifications).

Step 2: Evaluate feature based on Criteria

- Business Impact
  - How much value does the feature add to the business?
  - Does it improve distributor satisfaction, increase sales, or reduce operational costs?
- User Need
  - How critical is the feature for distributors?

- How many distributors are requesting it?

- Feasibility
  - How complex is the feature to implement?
  - Does it require significant changes to the ERP system or portal architecture?
  - What are the time and resource requirements?

Step 3: Prioritize Using the MoSCoW Method

Categorize the features into Must-Have**,** Should-Have**,** Could-Have, and Won't-Have based on the evaluation:

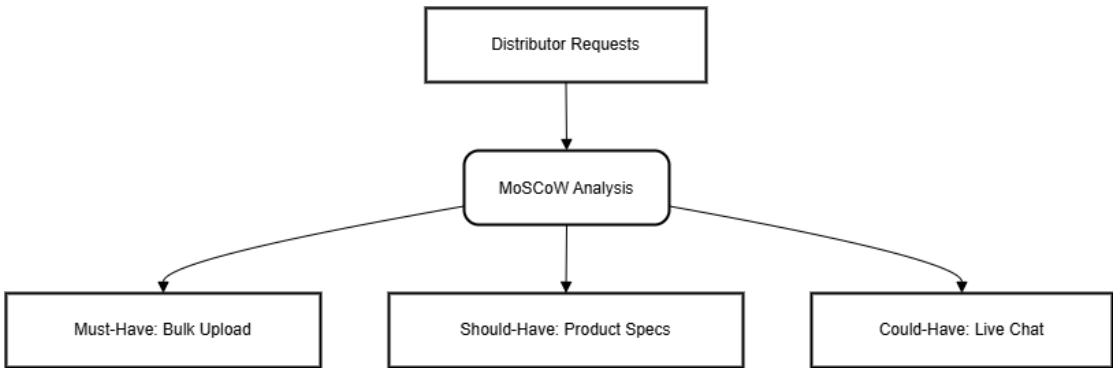
Feature Request	**Business Impact**	**User Need**	**Feasibility**	**Priority**	**Reasoning**
Bulk Order Upload	High	High	High	Must-Have	Essential for distributors managing large orders. Without this, order placement will be inefficient.
Detailed Product Specification	Medium	Medium	Medium	Should-Have	Enhances distributor decision-making but does not block core functionality.

Advanced Order Tracking	Low	Low	Medium	Could-Have	Useful but not critical.
Live Chat Support	Low	Low	Low	Won't- Have	A valuable enhancement but not crucial for launch. Can be considered in future updates.

Step 4: Communicate with stakeholders

- Inform distributors about which features will be available at launch and which will be added later
- Continuously collect feedback to refine priorities and ensure the portal meets distributor needs.

Representation of Feature prioritization:



TASK 2: Technical Aspect

a. Explain how you would ensure the accuracy and reliability of the two-way sync between the portal and the ERP system.

To ensure a seamless and error-free two-way synchronization between the B2B portal and Microsoft NAV ERP, I would follow a structured approach that combines technical best practices with project management strategies. Here's how I would achieve this:

1. Data Validation and Mapping

Goal: Ensure that data structures in both systems align perfectly to prevent inconsistencies.

Steps:

1. **Data Standardization**:
    - Define consistent data formats (e.g., date formats, product codes, pricing) across both systems.
    - Example: If the ERP system uses "DD/MM/YYYY" for dates, the portal should follow the same format.
2. **Data Mapping Rules**:
    - Map fields between the ERP and portal to ensure data flows correctly.
    - Example: Map the "Product Code" field in the ERP to the "SKU" field in the portal.
3. **Input Validation**:
    - Implement validation rules to catch errors like missing fields, invalid characters, or incorrect formats.
    - Example: Reject orders with missing customer details or invalid product codes.

2. API and Database Optimization

Goal: Prevent sync failures due to performance bottlenecks and ensure smooth data transfer.

Steps:

1. **Delta Sync Instead of Full Sync**:
    - Only transfer records that have changed since the last sync to reduce load.
    - Example: Sync only updated product prices instead of the entire product catalog.
2. **Pagination and Batching**:

- Sync large datasets in smaller chunks to avoid timeouts or system overload.
- Example: Sync 100 orders at a time instead of 10,000 in one go.

3. **Optimized Queries**:
    - Use indexed queries and efficient database calls to speed up data retrieval.
    - Example: Optimize SQL queries to fetch only necessary fields like product name and price.

3. Error Handling and Recovery Mechanisms

Goal: Detect, log, and resolve errors during data exchange to ensure data integrity.

Steps:

1. **Real-Time Error Logging**:
    - Log failed sync transactions for debugging and analysis.
    - Example: If an order fails to sync, log the error with details like timestamp and error code.
2. **Automatic Retry Mechanism**:
    - Automatically retry failed syncs after a short delay.
    - Example: If a sync fails due to a temporary network issue, retry after 5 minutes.
3. **Manual Sync Option**:
    - Allow administrators to manually trigger a sync if automatic syncs fail.
    - Example: Provide a "Sync Now" button in the admin panel for manual intervention.

4. Testing and Monitoring

Goal: Continuously test and monitor the sync process to detect and resolve issues early.

Steps:

1. **Unit Testing**:

- Test individual components like API endpoints and data transformations.
- Example: Verify that the API correctly fetches product details from the ERP.

2. **Integration Testing**:
   - Test the end-to-end sync process between the portal and ERP.
   - Example: Place an order in the portal and verify it appears correctly in the ERP.
3. **Load Testing**:
   - Simulate high traffic to ensure the sync process can handle large volumes of data.
   - Example: Test how the system performs when syncing 10,000 orders simultaneously.
4. **Audit Logs and Alerts**:
   - Monitor every sync event and send alerts for failures or anomalies.
   - Example: If stock levels in the ERP and portal differ, send an alert to the admin.

5. Security and Compliance

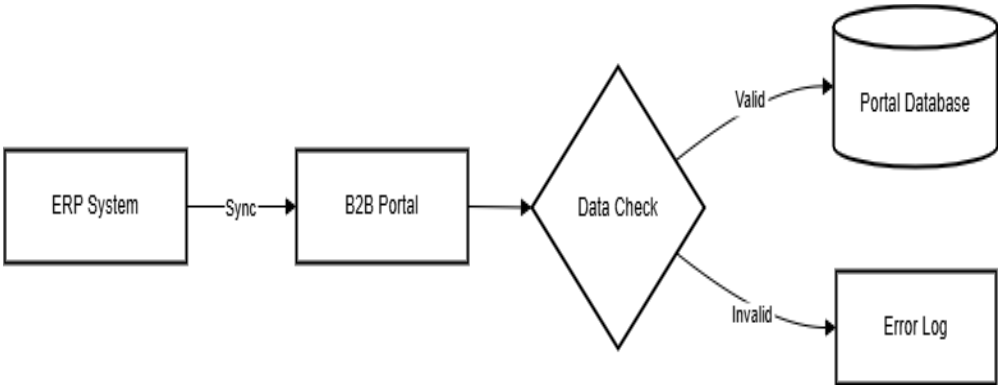
Goal: Protect data integrity and ensure compliance with industry standards.

Steps:

1. **Data Encryption**:
   - Use SSL/TLS to encrypt data in transit between the portal and ERP.
   - Example: Encrypt sensitive data like customer details and payment information.
2. **Access Control**:
   - Restrict sync access to authorized systems and users only.
   - Example: Use API keys or OAuth tokens to authenticate sync requests.
3. **Compliance Checks**:
   - Follow GDPR, ISO, or other industry-specific security protocols.

- Example: Ensure customer data is handled in compliance with GDPR regulations.

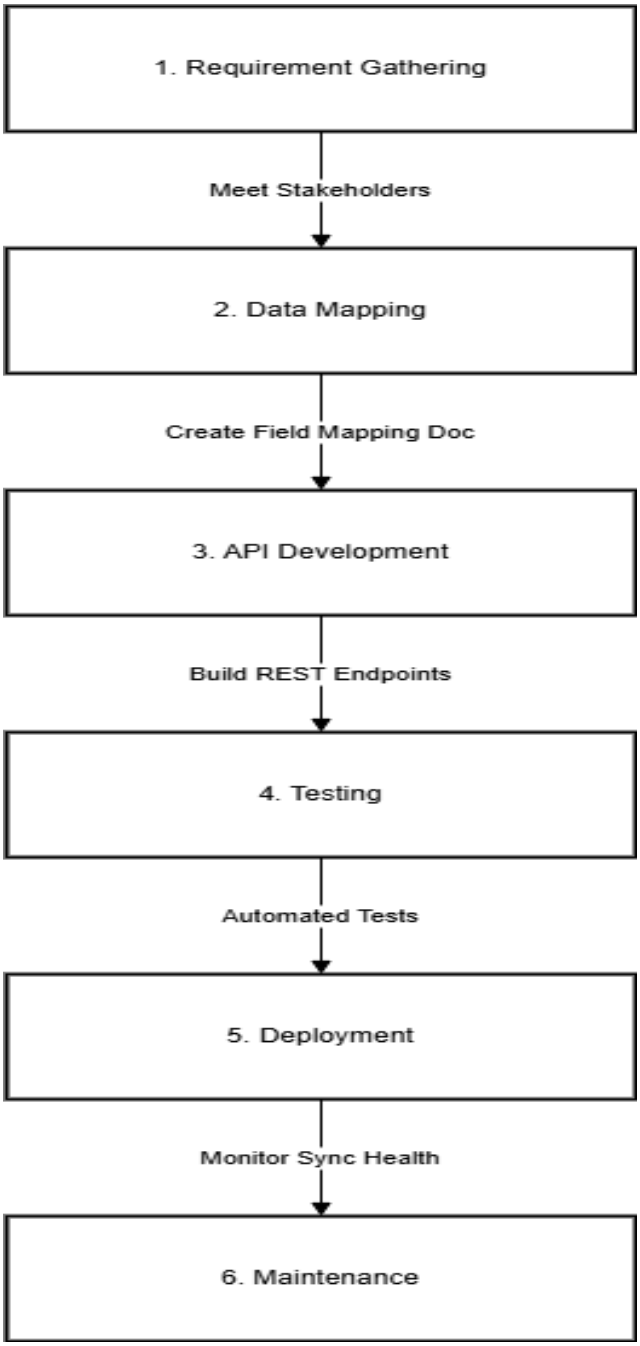
ERP Data Flow representation:



b. Highlight the technical workflow you would implement to address the integration and syncing process.

Here's how I would implement the integration and syncing process between the B2B portal and Microsoft NAV ERP using the 4DCX Framework:

Here's the technical workflow:



1. Requirement Gathering and Analysis

Goal: Define the integration scope and requirements.

- Stakeholder Collaboration: Work with the ERP team, developers, and distributors to identify the data fields (e.g., product details, orders, invoices) and workflows that need to be synced.
- Sync Rules Documentation: Define how data will flow between systems (e.g., which system is the source of truth for specific data).

- Dependency Identification: Identify dependencies, such as API availability, ERP system constraints, or third-party tools.

2. Data Mapping and Validation

Goal: Ensure data consistency and accuracy between systems.

- Data Mapping: Create a mapping document that aligns ERP fields with portal fields (e.g., ERP.ProductCode → Portal.SKU).
- Validation Rules: Implement validation logic to catch errors (e.g., reject orders with missing CustomerID or invalid ProductCode).
- Error Handling: Develop mechanisms to log and resolve sync errors (e.g., automatic retries for failed API calls or manual intervention for critical failures).

3. API and Database Design

Goal: Design a robust and scalable integration architecture.

- API Design: Define RESTful or GraphQL API endpoints for data exchange.

Example:

GET /products – Fetch product details from ERP.

POST /orders – Sync orders from the portal to ERP.

- Database Optimization:

Use indexed queries for faster data retrieval (e.g., CREATE INDEX idx_product_code ON Products(ProductCode)).

Implement pagination for large datasets (e.g., GET /products?page=1&limit=100).

- Delta Sync: Implement a mechanism to sync only changed records (e.g., use LastModifiedDate to track changes).

4. Development and Testing

Goal: Build and test the integration in iterative sprints.

- Sprint Planning: Break the integration into smaller tasks (e.g., product sync, order sync) and prioritize them using agile methodologies.
- Unit Testing: Test individual components (e.g., API endpoints, data transformation logic) using frameworks like JUnit or Mocha.

- Integration Testing: Test the end-to-end sync process between the portal and ERP system using tools like Postman or SoapUI.
- User Acceptance Testing (UAT): Involve distributors in testing to validate the sync process and ensure it meets their needs.

5. Monitoring and Optimization

Goal: Ensure the sync process is reliable and performs well under load.

- Real-Time Monitoring: Use monitoring tools like Prometheus or Datadog to track sync status, detect delays, and flag failures.
- Performance Optimization:
    - Optimize API calls (e.g., use caching with Redis to reduce redundant calls).
    - Optimize database queries (e.g., use EXPLAIN to analyze query performance).

- Audit Logs: Maintain logs of sync activities using tools like ELK Stack (Elasticsearch, Logstash, Kibana) for troubleshooting and auditing.

6. Deployment and Maintenance

Goal: Deploy the integration and ensure ongoing reliability.

- Staging Environment: Test the sync process in a staging environment that mirrors the production setup.

- Backup and Rollback Plan: Prepare a plan to revert changes if something goes wrong post-deployment (e.g., use database snapshots or versioned APIs).
- Continuous Improvement: Regularly review sync logs, gather feedback, and make improvements (e.g., optimize sync frequency based on usage patterns).

TASK 3: Management Aspect

a. How would you manage the delays caused by technical complexities and bring the project back on track?

As a Project Manager, addressing technical delays and maintaining clear communication with stakeholders necessitates a structured yet flexible strategy that incorporates Agile principles and modern project management technologies. Here's how I would approach the scenario as a project management, integrating Agile concepts and tools:

1. Identify the Root Cause of Delays

- Agile Retrospective: Conducting a retrospective with the development team to identify what is causing the delays (e.g., API integration challenges, data inconsistencies, or performance issues).
- RAID Log: Use the RAID Log (Risks, Assumptions, Issues, Dependencies) to document and track the technical complexities causing delays.
- Impact Analysis: Assess how the delays affect the project timeline, scope, and deliverables using tools like Jira or Trello.

2. Reassess the Project Plan

- MoSCoW Prioritization: Use the MoSCoW Method to reprioritize features:
    - Must-Have: Critical features like ERP integration and order placement.

    - Should-Have: Important but non-critical features like bulk order **upload.**
    - Could-Have: Nice-to-have features like advanced order tracking.
    - Won't-Have: Features like live chat support that can be deferred.
- Revised Timeline: Update the project timeline in Microsoft Project or Smartsheet to reflect realistic deadlines.
- Resource Reallocation: Use Resource Management Tools (e.g., Float, Resource Guru) to reallocate developers or QA engineers to high-priority tasks.

3. Implement Mitigation Strategies

- Break Down Tasks: Use Agile Sprints to break complex tasks into smaller, manageable sub-tasks. For example, divide the ERP integration into:
    - Data mapping.
    - API development.
    - Sync testing.
- Parallel Development: Run development and testing in parallel to save time. For example, while developers work on the API, QA engineers can test completed modules.
- Contingency Plans: Prepare backup plans for high-risk areas. For example, if the sync fails, have a manual data reconciliation process in place.

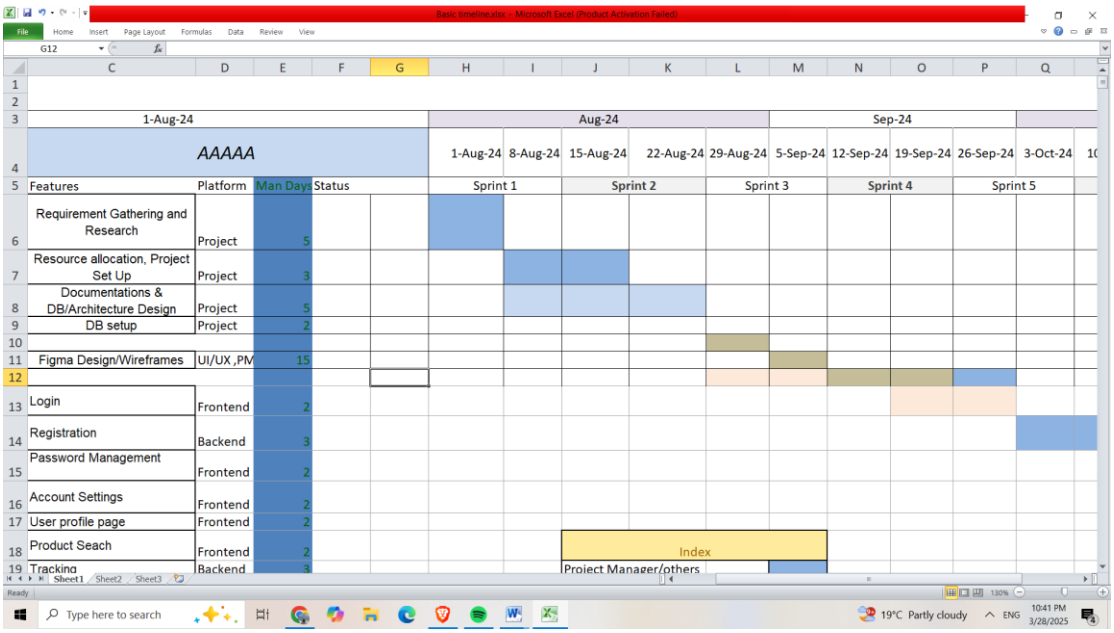
4. Monitor Progress Closely

- Daily Standups: Use Agile Standups to track progress and address blockers. Tools like Jira or Asana can help visualize task status.
- Sprint Reviews: Conduct sprint reviews at the end of each sprint to evaluate completed work and adjust priorities for the next sprint.
- Milestone Tracking: Use Gantt Charts in tools like Microsoft Project or ClickUp to track milestones and ensure timely delivery.

5.Communication with Stakeholders

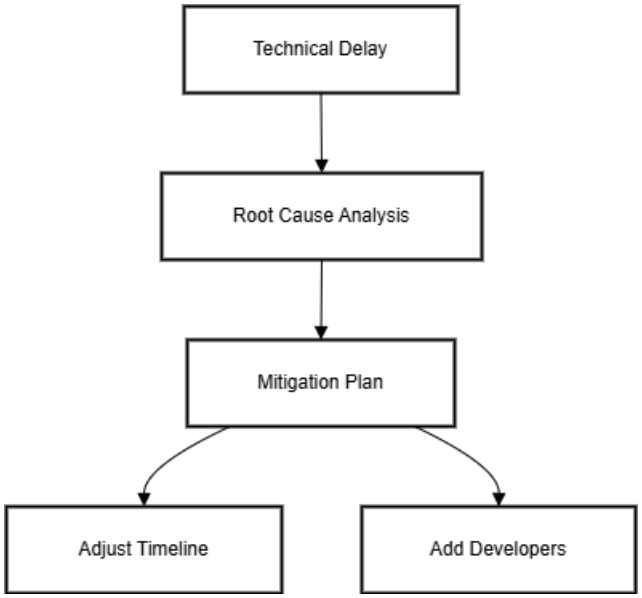
- Stakeholder Meetings: Scheduling meeting using zoom or Microsoft teams to discuss the revised plan and gather feedbacks.

Also I would Gantt chart to understand Further



	Features	Platform	Man Days	Status	Sprint 1	Sprint 2	Sprint 3	Sprint 4	Sprint 5
	1-Aug-24				Aug-24			Sep-24	
	AAAAA				1-Aug-24	8-Aug-24	15-Aug-24	22-Aug-24	29-Aug-24	5-Sep-24	12-Sep-24	19-Sep-24	26-Sep-24	3-Oct-24
6	Requirement Gathering and Research	Project	5											
7	Resource allocation, Project Set Up	Project	3											
8	Documentations & DB/Architecture Design	Project	5											
9	DB setup	Project	2											
10														
11	Figma Design/Wireframes	UI/UX ,PM	15											
12														
13	Login	Frontend	2											
14	Registration	Backend	3											
15	Password Management	Frontend	2											
16	Account Settings	Frontend	2											
17	User profile page	Frontend	2											
18	Product Seach	Frontend	2			Index								
19	Tracking	Backend	3			Project Manager/others								

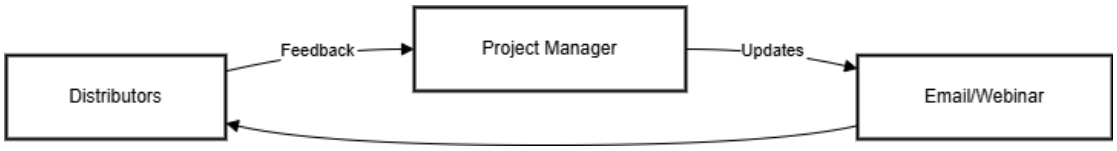
Delay Management Representation:



b. Outline a communication plan to inform the distributors about project progress and any potential changes to the timeline or features.

I will use an organized communication plan as the project manager to notify distributors on features, any modifications, and project status. This strategy minimizes interruptions to distributor operations while guaranteeing transparency, alignment, and stakeholder participation. Few of the objectives of communication plan are:

- Provide regular updates on project milestones and timelines.
- Manage expectations regarding feature additions, modifications, or delays.
- Ensure distributors' feedback is incorporated into decision-making.
- Maintain a clear escalation path for concerns and queries.



Timing	Channel	Audience	Content	Purpose
Weekly	Email Newsletter	All Distributors	Progress update: Completed features, current focus, and next steps.	Keep distributors informed about ongoing progress and upcoming milestones.
Bi-Weekly	Webinar / Demo	Technical Distributors	Live demo of new features (e.g., bulk order upload, ERP sync).	Provide hands-on understanding of new features and gather feedback.
Milestone Achieved	Portal Notification	All Distributors	Announcement: Key milestones	Celebrate progress and

			achieved (e.g., order sync feature now live).	inform distributors about new functionalities.
Ad Hoc	Email	Non-Technical Distributors	Notification: Delay in product sync due to technical complexities.	Communicate urgent updates or changes that impact distributors.
Ad Hoc	Webinar / Q&A	Technical Distributors	Deep dive into technical changes (e.g., API updates, sync improvements).	Address technical concerns and provide detailed explanations.
Monthly	Survey	All Distributors	Feedback survey: Rate the portal's usability and suggest improvements.	Gather feedback to improve the portal and address distributor pain points.
Post Launch	Email	All Distributors	Summary: Key features launched, next steps, and support resources.	Provide a wrap-up of the project and outline future enhancements.

4. User Stories

a. Create user stories for the portal's key functionalities, such as Registration, team creation, product search, order placement, and invoice viewing.

Here’s the link to the

Wireframe (Open with draw.io): Wireframe

User Stories in Trello: trello user stories