**Smart Farm-to-Market Management System**

A Salesforce-Powered Solution for the Modern Agricultural Ecosystem

Prepared By: Hariom Balang

**==================================================**

**Phase 1: Problem Understanding & Industry Analysis**

**Project Title: Smart Farm-to-Market Management System**

**Industry: Agriculture**

**Prepared By: [Hariom Balang]**

**1. Introduction**

The agricultural sector, particularly in rural areas, faces significant challenges in crop management, market access, subsidy utilization, pest control, transportation, and financial tracking. These processes are largely dependent on manual methods that result in inefficiencies, data inaccuracies, and communication gaps among stakeholders. The objective of this phase is to understand the underlying problems and define requirements for building a Salesforce-based solution that automates workflows, improves visibility, and enhances productivity — all without relying on external paid services.

This document outlines the activities performed in Phase 1, including requirement gathering, stakeholder analysis, business process mapping, industry-specific use case analysis, and exploration of Salesforce’s AppExchange offerings.

**2. Requirement Gathering**

The first step involved identifying the functional and non-functional requirements necessary to address the core problems in farm-to-market operations.

**Functional Requirements**

* **Crop Data Management**: Farmers should be able to enter crop details such as type, location, expected yield, and planting schedule.
* **Market Price Tracking**: Administrators should update market prices across regions, and farmers should receive notifications on price fluctuations.
* **Pest & Disease Alerts**: The system must allow for quick reporting of pest outbreaks and notify affected farmers instantly.
* **Subsidy Application Management**: Farmers should apply for subsidies via structured forms, with automated eligibility checks and approval processes.
* **Transport Scheduling**: Farmers should request transport services, and managers should assign agents based on availability and location.
* **Invoicing & Payment Tracking**: The system should generate invoices and track payments in real-time.
* **Reporting & Dashboards**: Users should access reports on crop production, pricing trends, subsidy utilization, and transport logistics.

**Non-Functional Requirements**

* **Usability**: The platform must be user-friendly, especially for farmers with limited technical experience.
* **Security**: Role-based permissions, data access controls, and privacy settings must be implemented.
* **Scalability**: The solution should support increased data volumes as more users and regions are added.
* **Performance**: The system should operate seamlessly, even with large datasets and concurrent users.
* **Accessibility**: Interfaces should be compatible with both desktop and mobile devices to accommodate users in remote areas.

**3. Stakeholder Analysis**

* **Farmers**
  + Primary users of the system
  + Responsibilities: Enter crop data, apply for subsidies, track crop status
  + Expectations: Simple interface, timely alerts, financial tracking
* **Co-operative Managers**
  + Supervisory users
  + Responsibilities: Manage market pricing, approve subsidies, assign transport
  + Expectations: Automated workflows, reporting dashboards, real-time monitoring
* **Transport Agents**
  + Service providers for logistics
  + Responsibilities: Handle scheduling and delivery of crops
  + Expectations: Efficient task allocation, clear tracking of deliveries
* **Government Bodies**
  + Policy administrators
  + Responsibilities: Approve subsidies, monitor compliance
  + Expectations: Structured data, accurate eligibility reporting, audit-ready information
* **Salesforce Admin**
  + System operator
  + Responsibilities: Configure platform, manage users, roles, and access
  + Expectations: Reliable, secure, and scalable environment for all users

**4. Business Process Mapping**

A detailed workflow was mapped to understand how data and tasks flow across stakeholders.

**Crop Planning Workflow**

* Farmers enter crop details → Co-operative manager reviews → Data stored for reporting.

**Market Price Update**

* Admin updates price trends → Notifications sent → Dashboard updated.

**Pest Alert Workflow**

* Pest outbreaks are reported → System identifies affected regions → Farmers are notified → Preventive actions advised.

**Subsidy Application Workflow**

* Farmer submits subsidy request → System validates eligibility → Manager reviews and approves → Farmer is notified.

**Transport Scheduling Workflow**

* Farmer requests transport → Manager assigns available agent → Delivery scheduled and tracked.

**Billing & Payment Workflow**

* Post-harvest, invoices are generated → Payments tracked → Reports updated for transparency.

These workflows help in identifying automation points that reduce manual effort and errors.

**5. Industry-specific Use Case Analysis**

A study of agricultural challenges and workflows highlighted the following:

**Challenges**

* Lack of structured data entry and storage.
* Inconsistent access to market information.
* Delayed communication between stakeholders.
* Manual subsidy applications prone to errors.
* Inefficient transport coordination leading to crop wastage.

**Best Practices**

* Centralized data systems that streamline crop tracking.
* Automated workflows to reduce manual intervention.
* Real-time alerts for pest management.
* Interactive dashboards to guide decision-making.

**Salesforce Alignment**

Salesforce’s standard functionalities like custom objects, flows, approval processes, and reporting tools provide a cost-effective solution to these challenges without relying on external paid APIs.

**6. AppExchange Exploration**

To ensure that the system leverages Salesforce’s ecosystem efficiently, an exploration of AppExchange was conducted.

**Findings**

* Several agriculture-related apps exist but mostly require paid subscriptions or external integrations.
* Free apps offered limited functionality or were not directly applicable to farm workflows.
* Salesforce’s native tools — custom objects, flows, validation rules, and dashboards — are sufficient to meet the project requirements.

**Phase 2: Org Setup & Configuration**

**1. Introduction**

In Phase 2, the Salesforce environment is configured to support the application’s business processes and workflows. This phase ensures that organizational settings, user permissions, roles, and data sharing rules are properly defined before implementation begins. The configuration focuses on scalability, data security, and user accessibility while using Salesforce’s free tools wherever possible.

This phase lays the foundation for building a robust and secure application tailored to the agricultural industry’s specific needs.

**2. Salesforce Editions**

For this project, the **Developer Edition Org** has been selected, as it provides access to all Salesforce features without subscription costs.

* **Key Features Available:**
  + Custom Objects, Fields, and Relationships.
  + Flows, Workflows, and Approval Processes.
  + Apex programming and Lightning Web Components (LWC).
  + Reports, Dashboards, and Analytics tools.
* **Limitations:**
  + Sandbox environment is not available in Developer Edition, so testing is performed in a separate environment or within the development org.

This edition ensures cost-efficiency while offering a fully functional Salesforce environment for development and testing.

**3. Company Profile Setup**

The organization’s profile was configured to align with regional requirements and agricultural workflows.

* **Steps Taken:**
  1. Accessed Setup → Company Information.
  2. Updated the organization’s name to “Smart Farm-to-Market Management System.”
  3. Set the default currency to **INR** to reflect local transactions.
  4. Set Locale to **India** for date formats and regional settings.
  5. Configured Timezone to **IST** (Indian Standard Time).
* **Purpose:**  
  These settings ensure that reports, dates, and financial fields align with real-world requirements, improving user experience and reducing confusion.

**4. Business Hours & Holidays**

Defining business hours and holidays is essential for scheduling transport services, subsidy approvals, and task assignments.

* **Business Hours Configured:**
  + Monday to Saturday, 9:00 AM to 6:00 PM.
* **Holidays Added:**
  + Republic Day
  + Independence Day
  + Diwali
  + Local harvest festival days
* **Use Case:**  
  Business hours ensure that automated processes such as task scheduling and approval routing occur within working hours. Holidays are considered to avoid assigning tasks when offices are closed.

**5. Fiscal Year Settings**

The fiscal year was set to align with agricultural cycles and reporting requirements.

* **Settings Applied:**
  + Fiscal year starts from **April 1st**, ending on **March 31st**, to correspond with the seasonal crop calendar.
* **Purpose:**  
  Aligning the fiscal year helps produce accurate seasonal reports and financial tracking for subsidies and crop payments.

**6. User Setup & Licenses**

Users were created to represent stakeholders in the system.

**Test Users:**

1. **Admin User** – Full access for system configuration and management.
2. **Co-operative Manager** – Oversees pricing, subsidy approvals, and transport coordination.
3. **Farmer User** – Inputs crop data, applies for subsidies, and requests transport.
4. **Transport Agent** – Manages transport logistics and delivery schedules.

* **Licenses Assigned:**
  + Standard Salesforce user licenses were used for all roles.
* **Purpose:**  
  Ensures role-based data access while allowing each user group to perform their respective tasks effectively.

**7. Profiles**

Profiles control baseline permissions for users based on their role.

**Profile Definitions:**

* **Admin Profile:**
  + Full access to all objects, fields, and reports.
  + Manage users and system settings.
* **Co-operative Manager Profile:**
  + Create, view, edit crop records, market prices, and subsidy applications.
  + Access approval workflows and reports.
* **Farmer Profile:**
  + Create and view crop entries.
  + Submit subsidy applications and transport requests.
  + View assigned invoices and notifications.
* **Transport Agent Profile:**
  + View assigned transport requests.
  + Update delivery status.
* **Use Case:**  
  Profiles ensure data privacy and structured access, enabling users to see only what’s relevant to their responsibilities.

**8. Roles**

A role hierarchy was designed to streamline data access based on organizational structure.

**Role Hierarchy:**

* **Admin → Co-operative Manager → Farmer → Transport Agent**
* **Benefits:**
  + Managers can view records submitted by their subordinates.
  + Farmers see only their own data.
  + Transport agents access requests assigned to them.

This hierarchy mirrors real-world communication flows and simplifies reporting.

**9. Permission Sets**

Permission sets were created to extend access for specific tasks without modifying existing profiles.

* Example Use Cases:
  + A farmer accessing new reports.
  + A manager using advanced dashboards during seasonal planning.

Permission sets ensure flexibility while maintaining security controls.

**10. Organization-Wide Defaults (OWD)**

The OWD was set to control data visibility at a global level.

**Configuration:**

* Crop Records → **Private**
* Market Prices → **Read Only**
* Subsidy Applications → **Private**
* Transport Requests → **Controlled by Parent**
* Invoices → **Private**

**Purpose:**  
These settings ensure that sensitive data such as crop details and subsidy requests are visible only to authorized users, enhancing privacy and compliance.

**11. Sharing Rules**

Sharing rules were implemented to allow data access based on user roles and business requirements.

**Example Rules:**

* Crop records shared with co-operative managers based on territory.
* Subsidy requests shared with approval managers.
* Transport requests visible to assigned transport agents.

**Purpose:**  
Facilitates data sharing where collaboration is required while ensuring restricted access elsewhere.

**12. Login Access Policies**

Default login policies were applied to ensure security while allowing authorized access.

* Two-factor authentication can be enforced.
* IP restrictions may be configured later based on regional needs.
* Login hours were defined to prevent access outside business hours if required.

**13. Dev Org Setup**

The project was built in Salesforce’s Developer Edition environment.

* All configurations and development work were performed in this org.
* Version control through local tools like VS Code and SFDX ensured backup and rollback capabilities.

**14. Sandbox Usage**

Since Developer Edition does not provide sandboxes, testing workflows, triggers, and reports were conducted in a dedicated testing environment.

* All changes were validated before deployment.
* Realistic sample data was created to simulate production scenarios.

**15. Deployment Basics**

For moving changes between environments and ensuring data integrity, the following tools were used:

* **Change Sets** – For transferring configuration settings.
* **Data Loader** – For importing bulk data such as farmers, crop records, and pricing information.
* **VS Code & Salesforce CLI (SFDX)** – For version-controlled deployment and automated script execution.

**Best Practices Followed:**

* Validate in sandbox/test environments before deployment.
* Create backup snapshots prior to applying updates.
* Document configurations for audit purposes.

**Phase 3: Data Modeling & Relationships**

Phase 3 of the **Smart Farm-to-Market Management System** focuses on designing the **data structure** and relationships in Salesforce. Effective data modeling ensures accurate storage, integrity, and accessibility of information for farmers, managers, and administrators.

**1. Standard & Custom Objects**

The system leverages **standard Salesforce objects** like User, Account, and Contact to manage users, farmer groups, and individual farmers. Standard objects also support reporting and dashboards for analytics.

In addition, **custom objects** were created to capture project-specific data:

* **Crop**: Records details for each crop planted, including crop type, season, expected yield, and farm location.
* **Market Price**: Stores the prices of crops across different markets and updates over time.
* **Pest Alert**: Captures notifications about pest or disease outbreaks affecting crops.
* **Subsidy Application**: Manages farmer applications for government subsidies and tracks approval workflows.
* **Transport Request**: Handles requests for crop delivery and assignment of transport agents.
* **Invoice**: Tracks crop sales, payments, and billing status.
* **Farmer Feedback**: Collects suggestions, complaints, or reports from farmers to improve system operations.

**2. Fields**

Each object contains fields designed for accuracy and usability. Field types include:

* **Text fields** for crop names, market names, and other short text entries.
* **Long Text Area** fields to store detailed descriptions, such as pest alert notes.
* **Number fields** for expected yields, quantities, and pricing calculations.
* **Date and DateTime fields** to track planting, harvest, or transport dates.
* **Picklist fields** for predefined options like crop type or subsidy status, ensuring data consistency.
* **Formula fields** to calculate derived values automatically, such as total invoice amounts.
* **Checkbox fields** to track binary statuses, such as approved or pending.
* **Lookup and Master-Detail fields** to establish relationships between objects, linking related records for workflows and reporting.

**3. Record Types**

Record Types allow different business processes and layouts within the same object. For example:

* The Crop object has record types for **Seasonal Crops** and **Perennial Crops**, providing tailored page layouts and validation rules.
* Subsidy Applications use record types for **State Subsidy** and **Central Subsidy**, enabling distinct approval processes and notifications for each type.

**4. Page Layouts**

Page layouts define which fields, sections, and actions appear to users. Examples include:

* The Crop layout displays crop details, farmer information, and expected yield.
* Subsidy Application layout shows subsidy type, requested amount, attached documents, and approval history.
* Invoice layout presents sold crops, quantities, payment status, and total amounts.

Page layouts ensure users see relevant information efficiently while minimizing clutter.

**5. Compact Layouts**

Compact layouts provide **quick views** for record highlights, especially useful on mobile devices. For instance, the Crop object’s compact layout displays the crop name, season, farmer, and status. This allows users to access critical information without opening the full record.

**6. Schema Builder**

The Schema Builder is a visual tool for designing and viewing the Salesforce data structure. It shows objects, fields, and relationships in a drag-and-drop interface. This tool helps admins and developers maintain a clear overview of the data model, simplifying design, troubleshooting, and communication.

**7. Lookup vs Master-Detail vs Hierarchical Relationships**

Different relationships define how records connect:

* **Lookup Relationship**: A loose link where child records can exist independently. For example, a Crop can be linked to Market Price, but deleting the Market Price does not delete the Crop.
* **Master-Detail Relationship**: A strong dependency where the child inherits the parent’s ownership and sharing settings. For example, a Subsidy Application depends on a Crop record; if the Crop is deleted, the Subsidy Application is also deleted.
* **Hierarchical Relationship**: Used exclusively on the User object to define managerial hierarchies, such as a Manager overseeing a group of farmers.

**8. Junction Objects**

Junction objects support **many-to-many relationships**. For example, crops can be assigned to multiple transport agents, and each transport agent can handle multiple crops. The junction object stores assignment details, such as pickup date and status, enabling flexible logistics management without duplicating records.

**9. External Objects**

External objects allow Salesforce to access data stored outside of the platform, such as government market databases. This ensures users can view and interact with external information in real-time without storing it in Salesforce, maintaining a seamless interface for farmers and managers.

**Phase 4: Business Logic & Process Automation**

**1. Introduction**

With the data model and security framework in place, Phase 4 focuses on implementing the core business logic of the Smart Farm-to-Market Management System. This phase translates the business process maps from Phase 1 into automated actions within Salesforce. The primary goal is to **reduce manual intervention, ensure data integrity, and provide timely notifications** to all stakeholders. By leveraging Salesforce's declarative automation tools like Flows and Approval Processes, we can build a responsive and efficient system that actively guides users through their tasks.

**2. Validation Rules**

Validation rules are the first line of defense for maintaining data quality. They prevent users from saving records with invalid or incomplete information.

* **Crop Object:**
  + A rule ensures the **Harvest Date** cannot be before the **Planting Date**.
  + A rule requires the **Expected Yield** to be a positive number greater than zero.
* **Subsidy Application Object:**
  + A rule prevents a farmer from submitting an application if the **Required Documents Attached** checkbox is not checked, ensuring all paperwork is complete.
  + A rule ensures the **Requested Amount** does not exceed a predefined limit based on the crop type or government policy.
* **Transport Request Object:**
  + A rule ensures the **Requested Pickup Date** is in the future and not on a company holiday.

**3. Formula Fields**

Formula fields perform real-time calculations based on other fields, providing users with instant insights without needing to run a report.

* **Crop Object:**
  + **Total Expected Value (INR)**: A currency formula that calculates Expected Yield \* Related Market Price. This gives farmers and managers a quick estimate of a crop's value.
  + **Days to Harvest**: A number formula that calculates Harvest Date - Planting Date to show the crop's growth cycle duration.
* **Invoice Object:**
  + **Amount Due**: A currency formula that calculates (Quantity Sold \* Price per Unit) - Amount Paid.
  + **Payment Status**: A text formula that displays "Paid," "Partially Paid," or "Unpaid" based on the Amount Due.

**4. Salesforce Flows**

Flows are the primary tool for automating complex business processes. We will use a combination of Record-Triggered, Screen, and Scheduled Flows to handle the system's core workflows.

* **Record-Triggered Flows (Run in the background):**
  + **Pest Alert Notification Flow:**
    - **Trigger:** A new Pest Alert record is created.
    - **Logic:** The flow gets the location from the Pest Alert, finds all Crop records in that same location, and creates a notification (e.g., a Task or a custom Notification record) for each affected farmer.
  + **Market Price Change Notification Flow:**
    - **Trigger:** A Market Price record is updated.
    - **Logic:** The flow sends a custom notification or an email alert to all farmers who have active Crop records of that specific type, informing them of the price fluctuation.
  + **Automated Invoice Generation Flow:**
    - **Trigger:** A Transport Request record's status is updated to "Delivered."
    - **Logic:** The flow automatically creates a new Invoice record, populating it with details from the related Crop and Farmer records.
* **Screen Flows (Guided user interaction):**
  + **Guided Subsidy Application:**
    - A multi-screen flow placed on the Farmer's homepage.
    - **Screen 1:** The farmer selects their Crop record.
    - **Screen 2:** The form displays relevant crop details and asks for the subsidy amount and required information.
    - **Screen 3:** Includes a file upload component for necessary documents.
    - **Final Action:** Creates the Subsidy Application record and automatically submits it for approval.
  + **New Transport Request Wizard:**
    - A simple, single-screen flow for farmers.
    - **Logic:** Allows the farmer to select their harvested crop, input the quantity, and choose a desired pickup date from a calendar. This simplifies the request process, improving usability.
* **Scheduled Flows (Run at specified times):**
  + **Weekly Manager Digest Flow:**
    - **Schedule:** Runs every Friday at 5:00 PM.
    - **Logic:** The flow gathers key metrics for the week (e.g., number of new crops planted, subsidies approved, transports completed) and sends a formatted summary email to all users with the "Co-operative Manager" role.

**5. Approval Processes**

Approval processes are used for workflows requiring formal sign-off from one or more users. The subsidy application is a perfect use case.

* **Subsidy Application Approval Process:**
  + **Entry Criteria:** Triggers when a Subsidy Application is created with a status of "Submitted."
  + **Initial Submitter:** The Farmer user.
  + **Step 1: Manager Review**
    - **Assigned Approver:** The Co-operative Manager linked to the farmer's region.
    - **Actions:** The manager can **Approve** or **Reject**. If rejected, the farmer is notified via email with the rejection reason.
  + **Step 2: Final Government Approval (Conditional)**
    - **Criteria:** This step runs only if the Requested Amount is greater than ₹50,000.
    - **Assigned Approver:** A user in the "Government Bodies" approval queue.
  + **Final Approval Actions:**
    - The record's status is updated to "Approved."
    - An email alert is sent to the farmer confirming the approval.
    - The record is locked to prevent further edits.
  + **Final Rejection Actions:**
    - The record's status is updated to "Rejected."
    - An email alert is sent to the farmer with the comments from the approver.

**6. Email Alerts & Templates**

Standardized email templates are created to ensure all automated communications are professional, clear, and consistent.

* **Subsidy Status Notification:** An email template used by the approval process to inform farmers if their application is approved or rejected.
* **Transport Confirmation:** An email sent to both the farmer and the assigned transport agent when a transport request is scheduled, containing details like date, time, and location.
* **Welcome Email:** A template used with a user creation flow to welcome new farmers to the platform and provide login instructions.

**Phase 5: Apex Programming (Developer)**

**1. Introduction**

While Salesforce Flow and Process Builder are powerful tools for automation, certain complex business requirements necessitate the use of Apex programming. This phase focuses on developing custom backend logic to handle large data volumes, implement sophisticated transaction control, and create highly specialized functionalities that are beyond the scope of declarative tools.

The goal of this phase is to build a robust, scalable, and efficient backend that supports the advanced operational needs of the Smart Farm-to-Market Management System, ensuring the platform can handle future growth and complexity without compromising performance. All code will be written with bulkification best practices in mind and will be accompanied by comprehensive test classes to ensure reliability and meet deployment standards.

**2. Apex Triggers**

Apex triggers are used to perform custom actions before or after records are inserted, updated, or deleted. They are essential for enforcing complex business rules and maintaining data integrity across related objects.

* **Use Case: Automated Inventory and Financial Roll-Ups**
  + **Trigger Name:** TransportRequestTrigger
  + **Object:** Transport Request
  + **Event:** after update
  + **Logic:**
    1. The trigger fires when a Transport Request record's **Status** field is updated to **"Delivered"**.
    2. It identifies the related Crop record and automatically updates its **Status** to **"Shipped to Market"**.
    3. Crucially, it then creates a new Invoice record, pre-populating it with the Farmer's details, the quantity from the Transport Request, and the current Market Price for that crop. This automates the entire post-delivery financial workflow.
* **Use Case: Aggregating Farmer Performance Metrics**
  + **Trigger Name:** InvoiceTrigger
  + **Object:** Invoice
  + **Event:** after update
  + **Logic:**
    1. The trigger fires when an Invoice record's **Payment Status** is changed to **"Paid"**.
    2. It retrieves the total paid amount from the invoice.
    3. It finds the related Farmer (Contact or Account record) and adds this amount to a custom roll-up summary field like **Total Revenue YTD** on the farmer's record. Since standard roll-ups don't work across lookup relationships, an Apex trigger is the ideal solution here.

**3. Batch Apex for Bulk Processing**

Batch Apex is used to process a large number of records asynchronously in smaller, manageable chunks, avoiding Salesforce governor limits. This is perfect for data maintenance and large-scale notification tasks.

* **Use Case: Mass Pest Alert Notifications**
  + **Class Name:** PestAlertNotifierBatch
  + **Functionality:** While a Flow can handle simple alerts, it struggles with outbreaks affecting thousands of farms. This batch class is designed for mass communication.
  + **Process:**
    1. **start() method:** Queries for all new Pest Alert records created in the last 24 hours.
    2. **execute() method:** For each Pest Alert, it runs a SOQL query to find all Crop records within the specified geographical area (e.g., by district or postal code). It then gathers a list of unique Farmer IDs associated with those crops.
    3. It creates a custom Notification record or a Task for each affected farmer, warning them of the pest outbreak and providing recommended actions.
    4. **finish() method:** Sends a summary email to the System Administrator detailing the number of alerts processed and notifications sent.
  + **Scheduling:** This class can be scheduled to run automatically every night using a Schedulable Apex class, ensuring timely alerts without manual intervention.

**4. Apex Controllers for Custom User Interfaces**

When building custom Lightning Web Components (LWC) for a rich user experience, Apex controllers are needed to fetch and process data from the server.

* **Use Case: Dynamic Transport Agent Assignment Dashboard**
  + **Class Name:** LogisticsController
  + **Functionality:** Provides the backend logic for an LWC dashboard used by Co-operative Managers to assign transport requests efficiently.
  + **@AuraEnabled Methods:**
    - getPendingRequests(String region): Returns a list of all Transport Request records with a "Pending" status in a specific region.
    - getAvailableAgents(String region, Date pickupDate): A complex method that queries for User records with the "Transport Agent" profile in the given region. It then cross-references this with existing Transport Request records to find agents who are **not** already scheduled for a pickup on the specified date.
    - assignAgent(Id requestId, Id agentId): An update method that assigns the selected agent to the transport request and changes its status to "Scheduled."

**5. SOQL (Salesforce Object Query Language)**

SOQL is used within Apex to read data from the Salesforce database. Advanced queries are necessary for the complex data retrieval logic required by our Apex controllers.

* **Example from LogisticsController:** The query to find available agents would be more complex than a standard Get Records in Flow.

SQL

// Find User IDs of agents who are already busy on a specific day

Set<Id> busyAgentIds = new Set<Id>();

for (Transport\_Request\_\_c tr : [SELECT Assigned\_Agent\_\_c FROM Transport\_Request\_\_c WHERE Pickup\_Date\_\_c = :pickupDate AND Status\_\_c = 'Scheduled']) {

busyAgentIds.add(tr.Assigned\_Agent\_\_c);

}

// Now, find all agents in the region who are NOT in the busy list

return [SELECT Id, Name FROM User WHERE Profile.Name = 'Transport Agent' AND Region\_\_c = :region AND IsActive = true AND Id NOT IN :busyAgentIds];

**6. Test Classes and Code Coverage**

To deploy Apex to a production environment, Salesforce requires that at least 75% of the code is covered by unit tests. Test classes are critical for ensuring code quality, reliability, and future maintainability.

* **Use Case: Testing the TransportRequestTrigger**
  + **Test Class Name:** TransportRequestTriggerTest
  + **@isTest Annotation:** The class is marked as a test class, so it doesn't count against org limits.
  + **Setup:**
    1. A test utility method (@testSetup) creates the necessary prerequisite data: a Farmer Account, a Contact, a Crop record, and a Transport Agent User.
    2. A test method creates a new Transport Request record with a "Pending" status.
  + **Execution:**
    1. Test.startTest() and Test.stopTest() are used to create a fresh set of governor limits.
    2. Inside these methods, the test updates the Transport Request status to "Delivered," causing the trigger to fire.
  + **Assertion:**
    1. The test class then re-queries the database to retrieve the updated Crop record and the newly created Invoice.
    2. System.assertEquals('Shipped to Market', updatedCrop.Status\_\_c, 'Crop status was not updated correctly.');
    3. System.assertNotEquals(null, newInvoice, 'Invoice was not created.');
    4. These assertions verify that the trigger's logic executed as expected.
  + **Bulk Testing:** The test methods will also test the trigger's logic with a list of 200 records to ensure it is properly bulkified and will not fail in a real-world scenario.

**Phase 6: User Interface (UI) Development**

**1. Introduction**

With the backend automation and data structures now in place, Phase 6 is dedicated to crafting the user experience. The primary audience for this system includes farmers who may have varying levels of technical literacy, so the interface must be exceptionally clear, intuitive, and accessible, especially on mobile devices.

This phase leverages the declarative power of the Lightning App Builder to construct the application's framework, navigation, and record pages. For more dynamic and interactive requirements, we will develop a custom Lightning Web Component (LWC) to create a sophisticated scheduling tool for managers. The goal is to build a user interface that minimizes clicks, surfaces the most relevant information, and guides users through their daily tasks effortlessly.

**2. Lightning App & Navigation Setup**

The foundation of the user experience is a dedicated Lightning App that consolidates all the necessary tools and objects into one place.

* **App Name:** Smart Farm-to-Market Management
* **Branding:**
  + A custom logo representing agriculture or technology will be uploaded.
  + The app's primary color will be set to a shade of green to align with the agricultural theme.
* **Navigation Tabs (Utility Bar):** Using the Lightning App Builder, we will configure the main navigation bar to provide one-click access to the most critical areas of the application:
  + **Home:** A landing page with role-specific dashboards.
  + **Crops:** The central hub for managing all crop records.
  + **Subsidy Applications:** A list view for farmers to track their applications and for managers to review pending ones.
  + **Transport Requests:** A queue for managing logistics.
  + **Invoices:** For tracking payments and financial records.
  + **Reports:** Access to all operational reports.
  + **Dashboards:** Visual overview of the system's performance.

**3. Custom Record Page Design**

Standard page layouts are often insufficient. We will use the Lightning App Builder to create dynamic, component-based record pages that are tailored to the needs of different users.

* **Farmer's Crop Record Page:**
  + **Highlights Panel:** Displays key fields at the top: Crop Name, Status, Season, and the Total Expected Value (INR) formula field.
  + **Path Component:** A visual representation of the crop's lifecycle: Planning -> Planted -> Growing -> Harvested -> Shipped to Market.
  + **Main Column (Left):** A "Details" tab containing all the fields for crop information.
  + **Side Column (Right):**
    - **Related List - Single Component:** A dedicated component showing only the Subsidy Applications related to this crop.
    - **Related List - Single Component:** Another component showing the Transport Requests for this crop.
    - This design ensures a farmer can see the entire history and all related activities for a specific crop on a single screen without needing to click away.
* **Co-operative Manager's Transport Request Page:**
  + **Highlights Panel:** Shows Request ID, Farmer Name, Status, and Requested Pickup Date.
  + **Two-Column Layout:**
    - **Left Column:** Contains the detailed information about the request (e.g., crop type, quantity, location).
    - **Right Column:** This is where we will embed our custom Lightning Web Component, the **Transport Scheduling Calendar**, providing an interactive, actionable tool right where the manager needs it most.

**4. Custom Lightning Web Component (LWC): Transport Scheduling Calendar**

This is the centerpiece of the custom UI development, designed to solve the complex task of assigning transport agents efficiently.

* **Component Name:** transportSchedulerLwc
* **Purpose:** To provide Co-operative Managers with a visual calendar interface to see which transport agents are available on specific days and to assign a pending request with a single click.
* **Placement:** This LWC will be embedded directly on the Transport Request Lightning record page.
* **Visual Design:**
  + The component will display a **weekly calendar view**.
  + **Rows:** Each row will represent an available Transport Agent (User).
  + **Columns:** Each column will represent a day of the week.
  + **Cells:** The cells will be color-coded to indicate availability:
    - **Green:** The agent is available on that day.
    - **Red:** The agent is already assigned to another transport request on that day.
    - **Gray:** Non-working day (e.g., Sunday, holidays).
* **Functionality & Interactivity:**
  1. **Data Retrieval:** When the component loads on a Transport Request page, it will automatically call the getAvailableAgents method from the LogisticsController Apex class (developed in Phase 5), passing the request's region and desired pickup date.
  2. **Assignment Logic:** When the manager clicks on a **green (available)** cell, the component will:
     + Capture the agentId from the row and the date from the column.
     + Call the assignAgent Apex method, passing the current recordId of the Transport Request, the selected agentId, and the date.
  3. **User Feedback:** Upon a successful assignment, a **toast notification** will appear at the top of the screen confirming, "Transport Agent [Agent Name] has been assigned successfully."
  4. **Data Refresh:** The component will then automatically refresh the data on the page to reflect the updated status and assigned agent on the Transport Request record.

**5. Mobile Experience and Accessibility**

A significant portion of users (farmers and transport agents) will access the system from the field.

* **Responsive Design:** All Lightning pages and components are designed to be responsive out-of-the-box, ensuring they function correctly on the Salesforce Mobile App.
* **Simplified Actions:** Quick Actions will be added to the mobile layout, allowing a farmer to **"Request Transport"** or **"Report Pest"** with a single tap from their phone.
* **Compact Layouts:** The compact layouts designed in Phase 3 will ensure that the most critical information is visible in the record headers on the mobile app, minimizing the need for scrolling. This provides farmers with immediate, at-a-glance information about their crops and requests.

**Phase 7: Integration & External Access**

**1. Introduction**

The Smart Farm-to-Market Management System now has a robust data model, powerful internal automation, and an intuitive user interface. Phase 7 focuses on breaking the boundaries of the Salesforce platform to communicate with external systems and empower users on the go.

The primary objective is to deliver critical, time-sensitive information directly to users via channels they already use, such as SMS, and to provide a fully-featured mobile experience. This phase ensures that farmers and transport agents, who are often in the field with limited access to desktops, can interact with the system seamlessly. We will achieve this by integrating with a third-party SMS service and by fully leveraging the capabilities of the Salesforce Mobile App.

**2. External API Integration for Real-Time Alerts**

While email alerts are useful, SMS notifications are far more effective for reaching farmers in rural areas where internet connectivity may be unreliable but cellular networks are prevalent. This requires integrating Salesforce with an external SMS gateway provider (e.g., Twilio, Plivo, or a local Indian provider).

* **Use Case: Instant Pest & Price Alerts**
  + **Pest Outbreak Alerts:** When a Pest Alert is created for a specific region, the PestAlertNotifierBatch class (from Phase 5) will be enhanced. In addition to creating internal notifications, it will make an Apex callout to the SMS API to send a text message to every affected farmer.
    - **Example SMS:** *"Pest Alert: Whitefly outbreak reported in your area. Please inspect your cotton crop immediately. For advice, contact your co-op."*
  + **Market Price Fluctuations:** The Market Price Change Notification Flow will be modified to trigger an Apex action that sends an SMS to subscribed farmers.
    - **Example SMS:** *"Market Update: The price for Grade A Wheat has increased to ₹2,150/quintal in the Nagpur market."*
* **Technical Implementation (Apex Callouts):**
  + An Apex method will be created to construct and send the HTTP request to the SMS provider's API endpoint.
  + To handle bulk notifications without hitting governor limits, these callouts will be made asynchronously using **@future(callout=true)** methods or, preferably, by adding a **Queueable Apex** job to the processing chain. This ensures the system remains responsive even when sending hundreds of text messages.

**3. Secure API Authentication with Named Credentials**

Hardcoding API keys, authentication tokens, or endpoint URLs directly into Apex code is a major security risk and a maintenance nightmare. **Named Credentials** solve this by separating authentication details from the code.

* **Setup Process:**
  1. Navigate to **Setup → Security → Named Credentials**.
  2. Create a new Named Credential, for example, named SMS\_Gateway\_API.
  3. **URL:** Enter the base URL of the SMS provider's API (e.g., https://api.twilio.com).
  4. **Authentication:** Configure the authentication protocol. For most APIs, this will be "Password Authentication," where the username is the Account SID/API Key and the password is the Auth Token. Salesforce encrypts and securely stores these secrets.
* **Benefit in Apex Code:**
  1. The Apex callout code becomes significantly cleaner and more secure. Instead of managing credentials, the code simply references the Named Credential.
  2. **Before (Bad Practice):**

Java

// Hardcoded URL and credentials

HttpRequest req = new HttpRequest();

req.setEndpoint('https://api.twilio.com/2010-04-01/Accounts/ACxxxxxxxx/Messages.json');

req.setHeader('Authorization', 'Basic ' + EncodingUtil.base64Encode(Blob.valueOf('ACxxxxxxxx:your\_auth\_token')));

* 1. **After (Best Practice with Named Credentials):**

Java

// Salesforce handles authentication automatically

HttpRequest req = new HttpRequest();

req.setEndpoint('callout:SMS\_Gateway\_API/2010-04-01/Accounts/ACxxxxxxxx/Messages.json');

// No need to set the Authorization header!

* 1. This approach allows administrators to update API keys or switch between sandbox and production endpoints without a single line of code change, enhancing security and agility.

**4. Full Mobile Access via the Salesforce App**

Providing mobile access is not just about viewing data; it's about enabling full job functionality for users in the field. The Salesforce Mobile App makes this possible with minimal extra development.

* **For Farmers:**
  + **Accessibility:** Farmers can view their Crop records, check the status of their Subsidy Applications, and see their Invoice history directly from their phones.
  + **Quick Actions:** We will create specific Quick Actions on the mobile app's home page and for the Contact object:
    - **"Submit Subsidy Application":** Launches the Screen Flow created in Phase 4, providing a guided, step-by-step process optimized for mobile.
    - **"Request Transport":** A simple action that creates a Transport Request record with a pre-filled status, making it a two-tap process.
* **For Transport Agents:**
  + **Task Management:** Agents can view a list of their assigned Transport Requests for the day.
  + **Offline Functionality:** Crucially, the Salesforce Mobile App's **caching and offline capabilities** will be enabled. An agent can download their daily schedule, travel to a remote farm with no internet, and update the status of a pickup to "In Transit." The app saves this change locally and automatically syncs it back to Salesforce as soon as a network connection is re-established.
  + **Push Notifications:** We will configure push notifications via the Notification Builder. When a manager assigns a new, urgent transport request, the agent will receive an immediate push notification on their phone, alerting them to the new task without needing to be in the app.

**Phase 8: Data Management & Deployment**

**1. Introduction**

With the application's logic, interface, and integrations fully developed, Phase 8 focuses on the meticulous processes of data management and deployment. This final phase ensures that the system is populated with accurate, clean data and that the entire configuration is moved from the development environment to the live production org in a structured and error-free manner.

The success of this phase is measured by the integrity of the initial data and the smoothness of the go-live transition. It involves selecting the right tools for data migration, establishing rules to maintain data quality, and executing a well-planned deployment strategy. 🚀

**2. Initial Data Population: Data Import Wizard**

For the initial, smaller-scale data loads managed by system administrators or co-operative managers, the Data Import Wizard is the ideal tool due to its user-friendly, step-by-step interface.

* **Use Case:** Uploading the first batch of Farmer records from a CSV file provided by a local co-operative.
* **Process:**
  1. **Prepare the Data:** Ensure the source spreadsheet has clearly labeled columns (e.g., "Farmer Name," "Mobile Number," "Village").
  2. **Launch the Wizard:** From Setup, navigate to the Data Import Wizard.
  3. **Choose Object:** Select the Account and Contact objects to import the farmer data. The wizard can handle both standard and custom objects.
  4. **Map Fields:** The tool provides an intuitive interface to map the columns from your CSV file to the corresponding fields in Salesforce (e.g., map "Mobile Number" to the Phone field).
  5. **Check for Duplicates:** The wizard has built-in functionality to prevent the creation of duplicate records based on fields like Name or Email.
* **Benefits:** It's perfect for non-technical users and for importing fewer than 50,000 records at a time.

**3. Bulk Data Migration: Data Loader**

For large-scale data migration, such as loading historical records or data from a legacy system, Data Loader is the necessary tool. It is a client application capable of handling millions of records.

* **Use Case:** Migrating several years of historical Crop and Invoice data from an existing database or a set of large spreadsheets.
* **Process:**
  1. **Install Data Loader:** Download and install the application from Setup.
  2. **Prepare Data Files:** Export the data into CSV files. Crucially, when dealing with related records (like loading Invoices that need to be linked to existing Farmers), you must first export the Farmer data from Salesforce to get their unique **Salesforce Record IDs**. This ID is then used in the Invoice.csv file to establish the correct relationship.
  3. **Perform Operations:** Log in to Data Loader and choose the operation (**Insert, Update, Upsert, Delete**). For the historical invoice data, you would use the "Insert" operation.
  4. **Field Mapping:** Map the CSV columns to the Salesforce object fields.
  5. **Execute and Monitor:** Start the process and monitor the success and error files generated by Data Loader. The error file is invaluable for troubleshooting any records that failed to load.
* **Benefits:** Essential for large data volumes, automated data loads, and complex data migration scenarios involving multiple related objects.

**4. Ensuring Data Quality: Duplicate & Matching Rules**

To prevent the creation of redundant records and maintain a clean database from day one, we will configure duplicate rules.

* **Use Case:** Preventing a user from creating a new Farmer (Contact) record if a farmer with the same mobile number or name and village already exists.
* **Implementation Steps:**
  1. **Create a Matching Rule:**
     + Go to **Setup → Matching Rules**.
     + Create a rule on the Contact object.
     + Define the matching criteria: Contact: Phone EXACT MatchBlank = FALSE. This means the rule will find duplicates if two contacts have the exact same phone number. A more complex rule could be (Contact: FirstName FUZZY Match) AND (Contact: LastName FUZZY Match) AND (Contact: Village\_\_c EXACT Match).
  2. **Create a Duplicate Rule:**
     + Go to **Setup → Duplicate Rules**.
     + Create a rule on the Contact object, using the Matching Rule created above.
     + **Action on Create/Edit:** Set the rule to **Block** the user from saving the duplicate record, displaying a custom error message like, *"A farmer with this mobile number already exists in the system. Please search for the existing record."*

**5. Deployment: From Development to Production**

Moving the application's configuration and code from the development org to the live production org is a critical step that requires precision.

* **Method 1: Change Sets (Declarative-Friendly)**
  + **Best for:** Projects with mostly declarative changes (objects, fields, flows, validation rules) and simpler Apex components.
  + **Process:**
    1. **Establish Connection:** In the production org, establish a deployment connection to the development org (Setup → Deployment Settings).
    2. **Create Outbound Change Set:** In the development org, create an "Outbound Change Set."
    3. **Add Components:** Manually add all the components you have built: Custom Objects (Crop, Invoice, etc.), Custom Fields, Apex Classes (LogisticsController), Apex Triggers, Lightning Web Components (transportSchedulerLwc), Profiles, Page Layouts, and Flows.
    4. **Upload:** Upload the Change Set to the production org.
    5. **Validate & Deploy:** In the production org, find the "Inbound Change Set." First, run a **validation** to check for any potential errors without committing the changes. If validation succeeds, click **Deploy**.
* **Method 2: Salesforce CLI (SFDX) & Version Control (Advanced)**
  + **Best for:** Complex projects with significant custom code, multiple developers, and the need for a repeatable, automated deployment process.
  + **Process:**
    1. **Setup a Project:** Create an SFDX project on your local machine and connect it to your development org.
    2. **Retrieve Metadata:** Use SFDX commands (sfdx force:source:retrieve) to pull all the metadata (objects, code, LWC, profiles) from the development org into your local project.
    3. **Version Control:** Commit these files to a version control system like **Git**. This creates a backup and a historical record of all changes.
    4. **Authorize Production Org:** Connect SFDX to your production org.
    5. **Deploy:** Use a single command (sfdx force:source:deploy) to push the metadata from your local project to the production org. You can run a validation-only deployment first to check for errors.
  + **Benefits:** This method is faster, more reliable for complex deployments, enables team collaboration, and provides a full audit trail of changes.

**Phase 9: Reporting, Dashboards & Security Review**

**1. Introduction**

With the application now fully built, populated with data, and accessible to users, this final phase focuses on unlocking the value within that data and hardening the system's security. Information without insight is just noise. This phase involves creating powerful reports and dashboards to give stakeholders like Co-operative Managers and Government Bodies a clear, real-time view of agricultural operations.

Simultaneously, we will conduct a comprehensive security review to validate all access controls, ensuring that sensitive farm and financial data is protected. This final check guarantees that the application is not only powerful but also secure and compliant before its official launch. 📊🔐

**2. Creating Insightful Reports**

Reports are the foundation of analytics in Salesforce. They allow users to slice and dice data to answer specific business questions. We will create a series of reports tailored to different user roles.

* **Report Type:** First, we will create a custom report type, such as **"Crops with Subsidy Applications and Invoices,"** to allow for analysis across these related objects in a single report.
* **Operational Reports (for Co-operative Managers):**
  + **Subsidy Approval Pipeline:** A summary report on Subsidy Application records, grouped by Status (Submitted, In Review, Approved, Rejected). This helps managers quickly identify bottlenecks in the approval process.
  + **Transport Logistics Overview:** A report on Transport Request records, grouped by Status and Assigned Agent. This gives a daily snapshot of the logistics workload and helps in managing agent assignments.
  + **Pending Invoices:** A simple tabular report listing all Invoice records where the Payment Status is "Unpaid" or "Partially Paid," sorted by the due date to prioritize collections.
* **Strategic Reports (for Senior Management/Government Bodies):**
  + **Subsidy Utilization by Region:** A matrix report showing the sum of approved subsidy amounts, with Crop Type as the rows and Farmer's Village/District as the columns. This provides a powerful view of how funds are being distributed geographically and across different crop sectors.
  + **Crop Yield vs. Market Price Trends:** A line chart report that tracks the average Expected Yield from Crop records against the average Market Price over the last 12 months, helping to analyze profitability and market dynamics.

**3. Building Action-Oriented Dashboards**

Dashboards are visual collections of reports that provide a high-level, at-a-glance overview of key business metrics. They are the primary tool for monitoring the health of the farm-to-market ecosystem.

* **Co-operative Manager Dashboard ("Mission Control"):**
  + **Component 1 (Gauge):** Transport Fulfillment Rate. This gauge will show the percentage of Transport Requests that have been completed (Status = "Delivered") against a goal of 95%.
  + **Component 2 (Bar Chart):** Pending Subsidy Applications by Age. This chart will show the number of open applications, bucketed by how long they have been pending (e.g., 0-7 days, 8-14 days, 15+ days), immediately highlighting aging requests that need attention.
  + **Component 3 (Table):** Top 10 Farmers by Revenue (YTD). This component will list the top-performing farmers based on the sum of their paid invoices.
  + **Component 4 (Donut Chart):** Crops Planted This Season. This chart will provide a visual breakdown of all active crops by type.
* **Government Body Dashboard ("Policy & Impact Overview"):**
  + **Component 1 (Map):** A map of the state/region showing the total subsidy amount disbursed per district, providing a clear visual of where funds are flowing.
  + **Component 2 (Metric):** Total Subsidy Funds Disbursed (Fiscal Year). A single, large number showing the total amount paid out.
  + **Component 3 (Line Chart):** Pest Outbreaks Reported by Month. This helps identify seasonal trends and plan for future preventive measures.

**4. Final Security & Compliance Review**

Before go-live, a thorough review of the security model is non-negotiable to protect sensitive user and financial data.

* **Review Organization-Wide Defaults (OWD):**
  + We will double-check that the OWD for sensitive objects like **Invoice** and **Subsidy Application** are set to **Private**. This ensures that, by default, a farmer can only see their own records, which is the foundation of our security model.
* **Review Profiles & Field-Level Security (FLS):**
  + This is a detailed, field-by-field audit. For example, we will verify that the **Farmer Profile** has **Read-Only** access to Market Price records—they can see prices but cannot change them.
  + We will also ensure that a field like Subsidy Rejection Reason on the Subsidy Application object is only visible to the Manager and the specific Farmer whose application it is, not to other farmers.
* **Enable and Review Audit Trails:**
  + **Field History Tracking:** We will enable this on critical objects and fields. For the Subsidy Application object, we will track history on the **Status** and **Approved Amount** fields. This creates a related list on the record showing who changed the field, when they changed it, and the old and new values, which is essential for accountability.
  + **Setup Audit Trail:** We will review the Setup Audit Trail to see a log of all administrative changes made during the project's development. This is useful for troubleshooting any last-minute configuration issues and confirms that no unauthorized changes were made.
  + **Login History:** We will review login history to ensure that all test logins were from expected locations and to establish a baseline for normal user activity before the system goes live.

**Phase 10: Final Presentation & Demo Day**

**1. Introduction**

Welcome to the finish line! 🏁 This phase is the culmination of all your planning, configuration, coding, and testing. The goal is to deliver a professional and compelling demonstration that showcases the end-to-end functionality of the Smart Farm-to-Market Management System.

This is your opportunity to narrate the project's journey: from identifying the core problems in the agricultural supply chain to building a powerful, integrated solution on the Salesforce platform. This phase focuses on structuring the presentation, scripting a live demo, and preparing the necessary documentation for project handoff and your professional portfolio.

**2. Structuring the Presentation (The Narrative)**

A great demo follows a clear story arc. Structure your presentation to guide your audience from the "why" to the "how" and finally to the "so what."

1. **The Problem (5%):** Briefly revisit the challenges from Phase 1. "Manual processes, delayed communication, and lack of data were causing significant inefficiencies for farmers and co-operatives."
2. **Our Solution (10%):** Introduce the "Smart Farm-to-Market Management System." "We built a centralized, mobile-first platform on Salesforce to connect farmers, managers, and transport agents, automating the entire process from crop planning to payment."
3. **Live Demonstration (50%):** This is the main event. Walk through a real-world scenario. (See the detailed script below).
4. **Technical Architecture (15%):** Briefly showcase the "under the hood" components. Display a simple slide showing the key pillars:
   * **Data Model:** Custom Objects
   * **Automation:** Flows, Approval Processes, Apex Triggers
   * **User Interface:** Lightning App Builder, Custom LWC (transportSchedulerLwc)
   * **Integration:** Apex Callouts to SMS API via Named Credentials
5. **Challenges & Solutions (10%):** Discuss one or two interesting challenges. "A key challenge was notifying thousands of farmers instantly about pest alerts without hitting Salesforce limits. We solved this using a scalable Batch Apex job."
6. **Business Impact & Future Scope (10%):** Conclude with the value proposition. "This system leads to faster payments for farmers, reduced crop wastage, and provides data-driven insights for managers. Future enhancements could include integrating with weather data APIs or IoT soil sensors."

**3. The Live Demonstration: A Day in the Life**

The demo should be a seamless, end-to-end flow. Use different browser windows or personas to switch between roles.

* **Scene 1: The Farmer (Priya) - On Her Mobile**
  1. **Show the Salesforce Mobile App Login:** "Let's start with Priya, a farmer in the field."
  2. **Request Transport:** She navigates to a Crop record that is ready for harvest. Using a **Quick Action** you created, she clicks **"Request Transport."**
  3. **Fill the Form:** A simple Screen Flow pops up, asking only for the quantity and desired pickup date. She submits it. "The interface is designed for simplicity and speed, requiring minimal data entry."
* **Scene 2: The Co-operative Manager (Rajesh) - At His Desk**
  1. **Show the "Mission Control" Dashboard:** "Now, let's switch to Rajesh, the co-op manager. On his dashboard, he immediately sees a new pending request in his 'Transport Logistics' component."
  2. **Open the Request:** He clicks into Priya's Transport Request record.
  3. **Use the Custom LWC:** "To assign this, Rajesh uses our custom **Transport Scheduling Calendar** right here on the page. He can visually see which agents are free. Sunil is available."
  4. **Assign with One Click:** He clicks on an available slot for Sunil. A toast notification confirms the assignment. "This one-click process replaces multiple phone calls and manual scheduling, saving significant time."
* **Scene 3: The Transport Agent (Sunil) - On the Road**
  1. **Show the Push Notification:** "Sunil, the transport agent, instantly receives a push notification on his phone about the new assignment."
  2. **Update Status:** After picking up the crop, Sunil opens the record in his Salesforce app and updates the status from "Scheduled" to "Delivered." "He can do this even in areas with poor connectivity, as the app will sync the update once he's back online."
* **Scene 4: The Automated Result - Back at the Office**
  1. **Show the Invoice:** "The moment Sunil updated the status, our Apex trigger fired. Let's refresh Priya's farmer record... and there it is. A new Invoice has been automatically generated, ready for the accounts team."
  2. **Show the Dashboard Update:** "Back on Rajesh's dashboard, the 'Transport Fulfillment Rate' gauge has updated in real-time, reflecting the completed task."
  3. **Mention the Integration:** "Simultaneously, an Apex callout sent Priya an SMS confirming that her crop was delivered and her invoice is ready. This closes the communication loop completely."

**4. Preparing Handoff & Portfolio Documentation**

After the presentation, provide materials that ensure the project's longevity and showcase your skills.

* **Handoff Documentation:**
  + **System Administrator Guide:** A technical document detailing all custom objects, fields, automation logic (with flow diagrams), Apex trigger handlers, and instructions for managing the SMS integration and Named Credentials.
  + **User Guides:** Create short, visual, role-based guides (PDFs or videos) for Farmers and Managers, explaining how to perform their top 3-5 tasks.
* **Portfolio Materials:**
  + **Project One-Pager:** A concise summary of the project. Include the problem statement, your solution, the tech stack used (Salesforce Platform, LWC, Apex, etc.), and the key business benefits achieved.
  + **Demo Recording:** Record your live demonstration. Edit it down to a polished 2-3 minute video. This is an incredibly powerful asset for your LinkedIn profile or personal portfolio website.
  + **Code on GitHub:** Sanitize and upload key pieces of your code—like the LogisticsController Apex class and the transportSchedulerLwc component—to a public GitHub repository. This demonstrates your coding proficiency to future employers.