# 🚀 Farmer Order Fulfillment Workflow - Detailed Test Specifications

## 📋 Overview

This document provides comprehensive, step-by-step test case descriptions for the Farmer Order Fulfillment workflow, emphasizing:

- **API interactions** (endpoints, request/response formats)
- **Expected HTTP responses** (status codes, response structures)
- **Database state changes** (specifically `Order_Items.item_status`)
- **Authentication & authorization** validation
- **Cross-system integration** testing

## 🧪 Test Scenario 1: Farmer Views and Updates Their Order Items Successfully

### Goal

Validate that a farmer can see only their relevant order items and update their status through the fulfillment stages (`pending` → `harvested` → `packed`).

### Pre-conditions

- A customer has successfully placed an order containing items supplied by a specific farmer
- The farmer is authenticated with valid JWT token
- Order_Item exists in database with default `item_status='pending'`

# Detailed Step-by-Step API Interactions

## SETUP PHASE

### Step 1: Customer & Admin Pre-setup

```
# Create comprehensive test data using existing fixtures
test_data = create_test_data(farmer_client, admin_client, ...)

# Expected Database State:
# - User (farmer) created with user_role='farmer'
# - Farm linked to farmer
# - Product and ProductListing created
# - Customer user created
# - Location and PaymentMethod created
```

### Step 2: Create Product Listing for Farmer A

```
POST /api/products/listings/
Authorization: Bearer {farmer_jwt_token}
Content-Type: application/json

{
    "farm": 1,
    "product": 1,
    "quantity_available": 20.0,
    "current_price": 150.0,
    "min_order_quantity": 1.0,
    "quality_grade": "premium",
    "is_organic_certified": true,
    "listing_status": "available",
    "notes": "Fresh harvest available"
}
```

### Expected Response:

```
HTTP/1.1 201 Created
Content-Type: application/json

{
    "listing_id": 1,
    "farm": 1,
    "product": 1,
    "quantity_available": 20.0,
    "current_price": "150.00",
    "listing_status": "available",
    "created_at": "2024-01-15T10:00:00Z"
}
```

### Database State Change:

- `ProductListing` record created with `listing_id=1`
- `quantity_available=20.0`

### Step 3: Customer Payment Method Setup

```
POST /api/payments/methods/
Authorization: Bearer {customer_jwt_token}
Content-Type: application/json

{
    "payment_type": "CashOnDelivery"
}
```

### Expected Response:

```
HTTP/1.1 201 Created
Content-Type: application/json

{
    "payment_method_id": 1,
    "payment_type": "CashOnDelivery",
    "is_active": true
}
```

### Step 4: Customer Adds Item to Cart

```
POST /api/carts/add_item/
Authorization: Bearer {customer_jwt_token}
Content-Type: application/json

{
    "listing_id": 1,
    "quantity": 5.0
}
```

**Expected Response:**

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "cart_id": 1,
    "items": [
        {
            "listing_id": 1,
            "quantity": 5.0,
            "price": "150.00",
            "subtotal": "750.00"
        }
    ],
    "total_amount": "750.00",
    "items_count": 1
}
```

**Database State Change:**

- `CartItem` created linking customer to ProductListing
- `quantity=5.0`

**Step 5: Customer Creates Order**

```
POST /api/orders/
Authorization: Bearer {customer_jwt_token}
Content-Type: application/json

{
    "delivery_location_id": 1,
    "payment_method_id": 1,
    "estimated_delivery_date": "2024-02-15",
    "delivery_time_slot": "morning",
    "special_instructions": "Please handle with care"
}
```

**Expected Response:**

```
HTTP/1.1 201 Created
Content-Type: application/json

{
    "order_id": 1,
    "order_status": "confirmed",
    "total_amount": "750.00",
    "estimated_delivery_date": "2024-02-15",
    "order_items": [
        {
            "order_item_id": 1,
            "listing_id": 1,
            "quantity": 5.0,
            "item_status": "pending",
            "farmer_id": 1
        }
    ]
}
```

**Database State Changes:**

- `Order` record created with `order_status='confirmed'`
- `OrderItem` record created with `item_status='pending'`
- `ProductListing.quantity_available` reduced by 5.0 (now 15.0)

# FARMER WORKFLOW TESTING

### Step 6: Farmer A Lists Their Order Items

```
GET /api/orders/farmer-items/
Authorization: Bearer {farmer_jwt_token}
```

### Alternative Endpoints to Test:

- `GET /api/order-items/my_items/`
- `GET /api/orders/items/` (with farmer filtering)

### Expected Response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "results": [
        {
            "order_item_id": 1,
            "order_id": 1,
            "listing_id": 1,
            "product_name": "Tomatoes",
            "quantity": 5.0,
            "item_status": "pending",
            "customer_name": "John Doe",
            "order_date": "2024-01-15T10:30:00Z",
            "estimated_delivery": "2024-02-15",
            "notes": null
        }
    ],
    "count": 1
}
```

### Security Validation:

- Response contains ONLY order items where `farmer_id` matches authenticated farmer
- No order items from other farmers visible

### Step 7: Farmer A Retrieves Specific Order Item Details

```
GET /api/orders/items/1/
Authorization: Bearer {farmer_jwt_token}
```

### Expected Response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "order_item_id": 1,
    "order_id": 1,
    "listing": {
        "listing_id": 1,
        "product_name": "Tomatoes",
        "farm_name": "Green Valley Farm"
    },
    "quantity": 5.0,
    "unit_price": "150.00",
    "subtotal": "750.00",
    "item_status": "pending",
    "customer": {
        "customer_id": 1,
        "name": "John Doe",
        "phone": "0701234567"
    },
    "delivery_location": {
        "address": "123 Main St, Nairobi",
        "delivery_instructions": "Please handle with care"
    },
    "order_date": "2024-01-15T10:30:00Z",
    "estimated_delivery": "2024-02-15",
    "notes": null,
    "created_at": "2024-01-15T10:30:00Z",
    "updated_at": "2024-01-15T10:30:00Z"
}
```

### Step 8: Farmer A Updates Status from 'pending' to 'harvested'

```
PATCH /api/orders/items/1/
Authorization: Bearer {farmer_jwt_token}
Content-Type: application/json

{
    "item_status": "harvested",
    "notes": "Items harvested and ready for processing"
}
```

**Expected Response:**

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "order_item_id": 1,
    "item_status": "harvested",
    "notes": "Items harvested and ready for processing",
    "updated_at": "2024-01-15T11:00:00Z",
    "status_history": [
        {
            "status": "pending",
            "timestamp": "2024-01-15T10:30:00Z"
        },
        {
            "status": "harvested",
            "timestamp": "2024-01-15T11:00:00Z",
            "notes": "Items harvested and ready for processing"
        }
    ]
}
```

**Database State Change:**

- `OrderItem.item_status = 'harvested'`
- `OrderItem.notes = 'Items harvested and ready for processing'`
- `OrderItem.updated_at` = current timestamp
- Optional: Status history/audit trail updated

**Step 9: Farmer A Updates Status from 'harvested' to 'packed'**

```
PATCH /api/orders/items/1/
Authorization: Bearer {farmer_jwt_token}
Content-Type: application/json

{
    "item_status": "packed",
    "notes": "Items packed and ready for delivery"
}
```

### Expected Response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "order_item_id": 1,
    "item_status": "packed",
    "notes": "Items packed and ready for delivery",
    "updated_at": "2024-01-15T12:00:00Z",
    "status_history": [
        {
            "status": "pending",
            "timestamp": "2024-01-15T10:30:00Z"
        },
        {
            "status": "harvested",
            "timestamp": "2024-01-15T11:00:00Z"
        },
        {
            "status": "packed",
            "timestamp": "2024-01-15T12:00:00Z",
            "notes": "Items packed and ready for delivery"
        }
    ]
}
```

### Database State Change:

- `OrderItem.item_status = 'packed'`
- `OrderItem.notes = 'Items packed and ready for delivery'`
- `OrderItem.updated_at` = current timestamp

### Step 10: Customer Verification of Status Updates

```
GET /api/orders/
Authorization: Bearer {customer_jwt_token}
```

### Expected Response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "results": [
        {
            "order_id": 1,
            "order_status": "confirmed",
            "order_items": [
                {
                    "order_item_id": 1,
                    "product_name": "Tomatoes",
                    "quantity": 5.0,
                    "item_status": "packed",  // Customer sees updated status
                    "farmer_name": "Jane Farmer",
                    "notes": "Items packed and ready for delivery"
                }
            ],
            "total_amount": "750.00",
            "estimated_delivery": "2024-02-15"
        }
    ]
}
```

### Cross-System Validation:

- Customer can see real-time status updates from farmer
- Order item status correctly reflects farmer's updates
- Data consistency maintained across Orders ↔ OrderItems

# 🔒 Test Scenario 2: Farmer Order Fulfillment -

# Unauthorized Actions

## Goal

Validate that a farmer cannot update order items they don't own (security testing).

## Pre-conditions

- Two farmers exist (Farmer A and Farmer B)
- An Order_Item belongs to Farmer A
- Farmer B is authenticated

## Detailed Step-by-Step Security Testing

### SETUP PHASE

#### Step 1: Create Farmer A with Order Item

```
# Farmer A setup (from previous scenario)
farmer_a_data = create_test_data(farmer_client, admin_client, ...)

# Farmer A has OrderItem with order_item_id=1, item_status='pending'
```

#### Step 2: Create and Authenticate Farmer B

```
POST /api/users/register/
Content-Type: application/json

{
    "phone_number": "0798123456",
    "password": "farmerb123",
    "re_password": "farmerb123",
    "first_name": "Bob",
    "last_name": "FarmerB",
    "user_role": "farmer",
    "email": "farmer.b@example.com"
}
```

### Expected Response:

```
HTTP/1.1 201 Created
```

### Login as Farmer B:

```
POST /api/users/jwt/create/
Content-Type: application/json

{
    "phone_number": "0798123456",
    "password": "farmerb123"
}
```

### Expected Response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "access": "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9...",
    "refresh": "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9..."
}
```

# SECURITY TESTING

### Step 3: Farmer B Attempts Unauthorized Update

```
PATCH /api/orders/items/1/
Authorization: Bearer {farmer_b_jwt_token}
Content-Type: application/json

{
    "item_status": "packed",
    "notes": "Unauthorized update attempt by Farmer B"
}
```

### Expected Response (Security Success):

```
HTTP/1.1 403 Forbidden
Content-Type: application/json

{
    "detail": "You do not have permission to perform this action.",
    "error_code": "permission_denied",
    "resource": "order_item",
    "resource_id": 1
}
```

**Alternative Expected Response:**

```
HTTP/1.1 404 Not Found
Content-Type: application/json

{
    "detail": "Not found.",
    "error_code": "resource_not_found"
}
```

**Security Analysis:**

- **403 Forbidden** = Resource exists but farmer lacks permission (explicit security)
- **404 Not Found** = Resource not visible to farmer (implicit security via filtering)
- Both responses indicate proper security implementation

**Database State Validation:**

- `OrderItem.item_status` remains `'pending'` (unchanged)
- `OrderItem.notes` unchanged
- No audit trail of unauthorized access attempt
- Farmer A's data integrity maintained

**Step 4: Farmer A Verifies Data Integrity**

```
GET /api/orders/items/1/
Authorization: Bearer {farmer_a_jwt_token}
```

**Expected Response:**

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "order_item_id": 1,
    "item_status": "pending",  // Still original status
    "notes": null,             // No unauthorized notes
    "updated_at": "2024-01-15T10:30:00Z"  // Original timestamp
}
```

### Step 5: Farmer B List Access Security Test

```
GET /api/orders/farmer-items/
Authorization: Bearer {farmer_b_jwt_token}
```

### Expected Response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "results": [],  // Empty - Farmer B has no order items
    "count": 0
}
```

### Security Validation:

- Farmer B sees empty list (no access to Farmer A's items)
- Proper role-based access control (RBAC) implementation
- Data isolation between farmers maintained

# 🔄 Test Scenario 3: Valid Status Transitions

# API Interactions for Status Transition Testing

### Test Transition: pending → harvested

```
PATCH /api/orders/items/1/
Authorization: Bearer {farmer_jwt_token}
Content-Type: application/json

{
    "item_status": "harvested"
}
```

### Expected Response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "order_item_id": 1,
    "item_status": "harvested",
    "updated_at": "2024-01-15T11:00:00Z"
}
```

### Test Transition: harvested → packed

```
PATCH /api/orders/items/1/
Authorization: Bearer {farmer_jwt_token}
Content-Type: application/json

{
    "item_status": "packed"
}
```

### Expected Response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "order_item_id": 1,
    "item_status": "packed",
    "updated_at": "2024-01-15T12:00:00Z"
}
```

# ❌ Test Scenario 4: Invalid Status Transitions

## API Interactions for Invalid Transition Testing

### Invalid Transition 1: pending → packed (skipping harvested)

```
PATCH /api/orders/items/1/
Authorization: Bearer {farmer_jwt_token}
Content-Type: application/json

{
    "item_status": "packed"
}
```

### Expected Response:

```
HTTP/1.1 400 Bad Request
Content-Type: application/json

{
    "detail": "Invalid status transition",
    "error_code": "invalid_transition",
    "current_status": "pending",
    "attempted_status": "packed",
    "valid_transitions": ["harvested"],
    "message": "Items must be harvested before they can be packed"
}
```

### Invalid Transition 2: packed → pending (backward)

```
# First set to harvested
PATCH /api/orders/items/1/
{
    "item_status": "harvested"
}

# Then try backward transition
PATCH /api/orders/items/1/
{
    "item_status": "pending"
}
```

**Expected Response:**

```
HTTP/1.1 400 Bad Request
Content-Type: application/json

{
    "detail": "Backward status transitions are not allowed",
    "error_code": "backward_transition",
    "current_status": "harvested",
    "attempted_status": "pending"
}
```

# 🔍 Test Scenario 5: Filtering and Search

## Status Filtering

```
GET /api/orders/farmer-items/?status=pending
Authorization: Bearer {farmer_jwt_token}
```

**Expected Response:**

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "results": [
        {
            "order_item_id": 1,
            "item_status": "pending",
            "product_name": "Tomatoes",
            "quantity": 5.0
        }
    ],
    "count": 1,
    "filters_applied": {
        "status": "pending"
    }
}
```

# Date Range Filtering

```
GET /api/orders/farmer-items/?date_from=2024-01-01&date_to=2024-12-31
Authorization: Bearer {farmer_jwt_token}
```

### Expected Response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "results": [...],
    "count": 5,
    "filters_applied": {
        "date_from": "2024-01-01",
        "date_to": "2024-12-31"
    }
}
```

# 📊 Integration Test Endpoints

## Farmer Dashboard Statistics

```
GET /api/orders/farmer-items/stats/
Authorization: Bearer {farmer_jwt_token}
```

**Expected Response:**

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "total_items": 15,
    "pending_items": 5,
    "harvested_items": 7,
    "packed_items": 3,
    "total_revenue": "5250.00",
    "recent_orders": 8
}
```

## Status-Specific Endpoints

```
GET /api/orders/farmer-items/pending/
GET /api/orders/farmer-items/harvested/
GET /api/orders/farmer-items/packed/
```

# 🎯 Key Database State Validations

## OrderItem Status Progression

```
-- Initial state
SELECT item_status FROM order_items WHERE order_item_id = 1;
-- Result: 'pending'

-- After harvested update
SELECT item_status, updated_at FROM order_items WHERE order_item_id = 1;
-- Result: 'harvested', '2024-01-15 11:00:00'

-- After packed update
SELECT item_status, updated_at FROM order_items WHERE order_item_id = 1;
-- Result: 'packed', '2024-01-15 12:00:00'
```

## Security Validation Queries

```
-- Verify farmer can only see their own items
SELECT COUNT(*) FROM order_items oi
JOIN product_listings pl ON oi.listing_id = pl.listing_id
JOIN farms f ON pl.farm_id = f.farm_id
WHERE f.farmer_id = {farmer_b_id};
-- Result: 0 (Farmer B has no items)

-- Verify unauthorized updates don't persist
SELECT item_status, notes FROM order_items WHERE order_item_id = 1;
-- Should NOT contain Farmer B's attempted changes
```

# 🔧 Implementation Notes

## Required Pytest Fixtures

- `customer_client` - Authenticated customer API client
- `farmer_client` - Authenticated farmer API client
- `admin_client` - Authenticated admin API client

- `db_reset` - Database cleanup between tests
- `create_test_data` - Comprehensive test data setup

## API Endpoint Patterns to Test

- `/api/orders/farmer-items/` (farmer-specific items)
- `/api/orders/items/{id}/` (specific item details/updates)
- `/api/order-items/my_items/` (alternative farmer items)
- `/api/orders/items/?farmer_id={id}` (filtered items)

## Error Handling Validations

- **401 Unauthorized** - Missing/invalid JWT token
- **403 Forbidden** - Valid token, insufficient permissions
- **404 Not Found** - Resource doesn't exist or not accessible
- **400 Bad Request** - Invalid status transitions, validation errors

This comprehensive test specification ensures thorough validation of the Farmer Order Fulfillment workflow with detailed API interactions, security testing, and database state management.