Project Proposal: PDMS for Microcrystalline Cellulose Plant

Client: Keerthika

Prepared By: Dinesh UNIQ

Date: 23-May-2025

---

1. **Objective**

- Replace paper-based records with a digital, secure data management system.
- Enable real-time dashboard visibility for plant managers.
- Detect and notify of anomalies or erroneous supervisor inputs.
- Provide detailed reports and analytics for operational decision-making.

System Architecture

- **Frontend:** React (Web)
- **Backend:** Django + Django REST Framework
- **Database:** PostgreSQL
- **Task Queue:** Celery + Redis
- **Notifications:** SMTP (Email), Twilio (SMS), WebSocket (In-App)
- **Storage:** AWS S3 (for documents/reports)

Modules

Module 1: User & Access Management

- Role-based login: Supervisor, QA, Manager, Admin
- Permissions for viewing, editing, and reporting

Module 2: Material Management

- Vendor records
- Raw material stock entries
- Incoming quality check

Module 3: Production Entry

- Batch entries by supervisors

- Timestamp, material lot tracking
- Shift-wise process records

Module 4: Quality Control

- Parameter definitions
- Specification range checks
- QC result logging and violation alerts

Module 5: Inventory & Warehouse Tracking

- Real-time stock updates
- Material issuance, usage, returns
- Threshold and reorder alerts

Module 6: Anomaly Detection Engine

- Rule-based input validation
- Anomaly scoring
- Flag high-risk patterns (e.g. repeated mistakes)

Module 7: Shift Management

- Shift schedule entries
- Supervisor assignments
- Auto alert on missing entries

Module 8: Audit & Logs

- CRUD logs
- Record lifecycle tracking
- Admin access for audits

Module 9: Notification & Escalation System

- Real-time alerts (QC fail, stock low, data anomaly)
- Escalation tree config (email/SMS)
- Notification center UI

Module 10: Reporting & Analytics

- Standard reports (Shift Summary, QC Trends, Stock Usage)
- Ad-hoc filters

- Export to PDF/Excel

Process Flow

1. Supervisor logs into system and enters batch/material data
2. QC is notified of new batch and enters result
3. Anomaly detection flags any invalid entries
4. Reports auto-generate daily and are emailed to managers
5. Notifications escalate if anomalies or unclosed shifts persist

---

Key Features

- Real-time dashboard for plant overview
- Validation engine for minimizing human error
- Multi-channel alerts (in-app, email, SMS)
- Scheduled report delivery
- Secure access with full audit traceability

Tech Stack Summary

- **Frontend:** React + TailwindCSS + Axios + Chart.js
- **Backend:** Django, DRF, Celery
- **Database:** PostgreSQL, Redis
- **Storage:** AWS S3 for media/report archival
- **APIs:** RESTful services with token-based auth

Deployment Strategy

- CI/CD via GitHub Actions
- Dockerized application stack
- Nginx + Gunicorn deployment on AWS EC2
- PostgreSQL on RDS, Redis on Elasticache
- Centralized logs via CloudWatch

Roadmap & Enhancements (Post-Phase 1)

- Mobile version (PWA or native)
- AI-based anomaly prediction
- Integration with SAP or other ERP systems
- Biometric supervisor login
- Offline data entry with sync

Conclusion

The PDMS project will be a foundational step in modernizing a high-impact manufacturing unit. It will drastically improve traceability, efficiency, and operational clarity while enabling proactive anomaly management. The flexible, modular architecture ensures the system is scalable and future-ready for extensions like AI or IoT integrations.

---

2. Core Features

2.1. Data Digitization & Entry

- Supervisor entry panels (mobile/tablet/PC compatible)
- Shift-wise input forms with validation
- Batch and process logs: Raw Material → Final Product

2.2. Stage-wise Process Monitoring

- Logs for: Pre-treatment, Drying, Grinding, Sieving, Blending
- Input/output material reconciliation
- Machine ID, operator, downtime logs

2.3. Quality Control Logging

- Parameter-specific test logs (Moisture, Density, Granule Size, etc.)
- In-process & final product checks
- Flagged re-tests and QC audit trails

2.4. Anomaly Detection System

- Rule-based + ML-enabled detection of data inconsistencies
- Auto-alerts for input errors, misaligned quantities, repeated timestamps

- Manager-level dashboard for review and escalation

2.5. Managerial Dashboard

- Real-time plant overview
- Batch journey tracing
- Shift efficiency KPIs, rejections, downtime trends
- Anomaly heatmaps and reports

2.6. Traceability & Reporting

- Raw → Stage → Product → Dispatch mapping
- Auto-generated reports (Excel/PDF)
- Scan & upload shift cards, test reports, GRNs

---

3. Technology Stack

Component	Stack
Backend	Django, Django REST Framework
Frontend	React (or Vue), Tailwind CSS
Database	PostgreSQL
Async/Workers	Celery + Redis
Anomaly Engine	Rule Engine + Scikit-learn (Phase 2)
File Storage	AWS S3 or Local FileSystem
Deployment	Dockerized app, CI/CD ready

---

4. Modules Breakdown

1. User & Role Management
2. Raw Material Management
3. Production Process Logging
4. Quality Control Module
5. Final Product & Dispatch
6. Anomaly Detection Engine

7. Dashboard & Reports
8. Document Storage & Audit Trail

PROCESS FLOW – PLANT CYCLE

1. Raw Material Intake

- Who: Store Supervisor
- What Happens:
    - Logs incoming vendor supply (name, quantity, batch ID)
    - Performs basic quality check (moisture %, foreign particles)
    - Assigns storage location
- System Needs:
    - Material master & vendor master
    - Real-time batch tracking
    - QC flag before allowing material use
    - Paper-to-digital sync option (photo/scan of GRN)

---

2. Material Allocation to Production

- Who: Shift Supervisor
- What Happens:
    - Picks batch for use (from inventory)
    - Logs stage input (e.g., Pre-treatment input: 500kg from Batch A1)
- System Needs:
    - Inventory lookup (what's available, approved)
    - Time-stamped, user-attributed entries
    - Input-Output linkage for traceability
    - Editable only within same shift, then lock

---

3. Stage-wise Processing

- Who: Supervisor

- What Happens:
    - At each stage: Drying → Grinding → Sieving → Blending
    - Logs input from previous stage
    - Captures machine ID, operator name, time in/out
    - Notes downtime or unusual events
- System Needs:
    - Dynamic stage form
    - Automatic stage chaining (batch ID + output = next input)
    - Conditional validations (e.g., drying temp must be > X°C)
    - In-line anomaly feedback (flag when loss > threshold)

---

4. In-Process Quality Checks

- Who: QA Analyst
- What Happens:
    - Samples taken per batch
    - Test results entered: moisture, bulk density, particle size
    - Approval/rejection flag
- System Needs:
    - Test parameter engine per stage
    - Auto-fail logic (e.g., granule size > limit → block next stage)
    - Re-sample tracking
    - Manual override with reason + approval path

---

5. Final Product & Packing

- Who: Supervisor & Dispatch Clerk
- What Happens:
    - Final yield logged
    - Packed in defined SKUs
    - Dispatch records created (invoice, vehicle, customer)

- System Needs:
    - SKU master
    - Packaging-to-batch linkage
    - Auto-generated batch dispatch report
    - Export-ready COA file attachment

---

6. Anomaly Detection

- Who: System + QA + Manager
- What Happens:
    - System checks for unusual data entries (e.g., <10% yield)
    - Flags for review
    - Supervisor or QA can justify or escalate
- System Needs:
    - Rule engine: static (thresholds), dynamic (pattern-based)
    - Feedback loop to learn from false positives
    - Dashboard widget: “Open anomalies by stage/by shift”
    - Alert mechanism: in-app + email/SMS

---

7. Managerial Dashboard

- Who: Plant Manager, QA Head
- What Happens:
    - View real-time status (current running batch, yields, downtimes)
    - Review historical performance (weekly trends, vendor rejection rate)
    - Approve/reject flagged anomalies
- System Needs:
    - KPI widgets: batch efficiency, rejection %, downtime
    - Drill-downs: batch > stage > operator
    - Download/export options
    - Anomaly heatmaps

Module 1: User & Role Management

Requirement Analysis (Functional)

Area	Requirements
Authentication	JWT/Session-based, role-based UI
Data Entry	Offline-first, shift-bound edit windows, form auto-validations
Batch Tracking	Full journey from raw → stage → product → dispatch
Stage Validation	Material reconciliation, input/output ratio, time limits
Anomaly Engine	Real-time rules + async job-based ML check
Notifications	Triggered on anomalies, escalated after timeout, delivery logs
Document Handling	Scanned shift card uploads, QC reports, auto-linked to batch
Reports & Audit	Change history, download logs, time/user of last modification

---

Requirement Analysis (Non-Functional)

NFR	Details
Availability	99.9% uptime; ideally cloud-hosted or local with offline fallback
Security	Role separation, activity logging, audit trails, data integrity checks
Scalability	Must support future integration (weigh bridges, IoT devices)
Performance	Real-time alerts within <10s of anomaly detection
Compliance	21 CFR Part 11 style logging optional (digital signature, versioning)

Module 1: User & Role Management

---

1. High-Level Design (HLD)

Objective:

Manage plant users with role-based access control (RBAC), shift-based data visibility, and audit trails.

Key Features:

- User registration & activation (by Admin only)
- Role-based access:
    - Admin: Full access, user mgmt
    - Supervisor: Data entry (Shift-bound)
    - QA: QC entry + override anomalies
    - Manager: Dashboard + anomaly approvals
- Shift assignment & lock control
- Login/authentication (JWT-based)
- Session timeout and activity log

---

2. Low-Level Design (LLD)

2.1. Database Models

```
# models.py

class Role(models.Model):
    name = models.CharField(max_length=20, unique=True)  # Admin, Supervisor, QA, Manager

class User(AbstractBaseUser):
    email = models.EmailField(unique=True)
    full_name = models.CharField(max_length=100)
    role = models.ForeignKey(Role, on_delete=models.PROTECT)
    is_active = models.BooleanField(default=True)
    shift = models.CharField(max_length=10, choices=[("A", "Shift A"), ("B", "Shift B"), ("C", "Shift C")])
    last_login = models.DateTimeField(null=True, blank=True)

class UserSessionLog(models.Model):
    user = models.ForeignKey(User, on_delete=models.CASCADE)
```

```
    login_time = models.DateTimeField(auto_now_add=True)
    logout_time = models.DateTimeField(null=True, blank=True)
    ip_address = models.GenericIPAddressField(null=True, blank=True)
    user_agent = models.TextField(null=True, blank=True)
```

2.2. API Endpoints (Django DRF)

Endpoint	Method	Access	Description
/api/auth/login/	POST	Public	JWT token response
/api/auth/logout/	POST	Authenticated	Token invalidation
/api/users/	GET/POST	Admin only	Create/read user
/api/users/<id>/	GET/PUT/DELETE	Admin only	Manage user
/api/profile/	GET	Authenticated	Get logged-in user details

2.3. Role-Based Middleware

```
# middleware.py

class RoleAccessMiddleware:
    def __init__(self, get_response):
        self.get_response = get_response

    def __call__(self, request):
        role = getattr(request.user, 'role', None)
        path = request.path
        if path.startswith('/api/admin/') and role.name != 'Admin':
            return JsonResponse({'error': 'Unauthorized'}, status=403)
        return self.get_response(request)
```

2.4. Business Rules

- A user cannot log data outside their assigned shift (check current server time vs shift slot)
- User creation only by Admin
- Only Admin can reassign shift
- Deleted users are soft-deleted (is_active=False)
- Login attempts logged with IP & user agent

2.5. UI Components (React)

Page	Components
Login	LoginForm, ErrorBanner
Profile	InfoCard, ChangePassword
User Management (Admin only)	UserTable, AddUserModal, RoleDropdown, ShiftSelector

---

2.6. Validation

- Email format validation
- No duplicate email
- Password: Min 8 chars, 1 uppercase, 1 special char
- JWT expiration handling (refresh token)

2.7. Sample Use Case

1. Admin creates Supervisor “Ramesh”, assigns Shift A.
2. Ramesh logs in at 6:00 AM (Shift A start), token issued.
3. He tries accessing Shift B logs → rejected with error.
4. At 2:00 PM, system auto-locks Shift A entries.
5. Manager views user activity logs for audit.

Module 2: Raw Material Management

---

1. High-Level Design (HLD)

Objective:

Digitally track all inbound raw material stock entries from vendors, capture quality checks, maintain inventory status, and ensure traceable input linkage to production batches.

Key Features:

- Vendor master and raw material catalog
- Goods Receipt logging (GRN)
- Basic QC on receipt (moisture %, impurities, etc.)
- Approved vs. Quarantined stock tracking
- Real-time inventory visibility
- Batch-to-stock allocation during production

---

2. Low-Level Design (LLD)

2.1. Database Models

```
# models.py

class Vendor(models.Model):
    name = models.CharField(max_length=100)
    contact_person = models.CharField(max_length=100)
    contact_email = models.EmailField()
    gst_number = models.CharField(max_length=15)

class RawMaterial(models.Model):
    name = models.CharField(max_length=100)
    code = models.CharField(max_length=10, unique=True)
    unit = models.CharField(max_length=10)  # e.g., KG, Litre
```

```
class MaterialBatch(models.Model):
    raw_material = models.ForeignKey(RawMaterial, on_delete=models.CASCADE)
    batch_code = models.CharField(max_length=50, unique=True)
    vendor = models.ForeignKey(Vendor, on_delete=models.SET_NULL, null=True)
    quantity_received = models.DecimalField(max_digits=10, decimal_places=2)
    quantity_available = models.DecimalField(max_digits=10, decimal_places=2)
    received_date = models.DateField()
    qc_status = models.CharField(max_length=10, choices=[('PENDING', 'Pending'), ('PASS', 'Pass'), ('FAIL', 'Fail')])
    remarks = models.TextField(null=True, blank=True)
```

2.2. API Endpoints (Django DRF)

Endpoint	Method	Access	Description
/api/vendors/	GET/POST	Admin	Manage vendors
/api/raw-materials/	GET/POST	Admin	Add/view materials
/api/material-batches/	GET/POST	Supervisor	Log received stock
/api/material-batches/<id>/qc/	POST	QA	Update QC status
/api/inventory/	GET	Supervisor/Manager	View current inventory

---

2.3. Business Rules

- Duplicate batch codes are not allowed.
- Quantity must be greater than zero.
- Only material batches with qc_status="PASS" can be allocated to production.
- Upon allocation, quantity_available is decremented automatically.
- Material expiry logic (if applicable) can be added.

2.4. UI Components (React)

Page	Components
Vendor Mgmt	VendorTable, AddVendorForm
Raw Material Catalog	MaterialList, AddMaterialModal
GRN Entry	MaterialBatchForm, AutoBatchCodeGen
QC Update	QCChecklistForm, StatusToggle
Inventory View	InventoryTable, FilterByMaterial

2.5. Inventory Calculation Logic

```
def get_current_inventory():
    return MaterialBatch.objects.filter(
        qc_status='PASS',
        quantity_available__gt=0
    ).values('raw_material__name').annotate(
        total_available=Sum('quantity_available')
    )
```

2.6. Validation & Security

- Supervisor can only view/add batches for their shift
- Admin can see all vendors/materials
- QC form locked after 24 hours (editable only by QA)
- Any manual override is logged with user + timestamp

2.7. Sample Workflow

1. Store Supervisor logs:
    - 800kg of "Alpha Cellulose" from Vendor X, Batch ALC-24-051.

2. QC Analyst logs test:
    - Moisture: 4.2%, Foreign Matter: None → Status: PASS.
3. Inventory auto-updates:
    - 800kg available in inventory under ALC-24-051.
4. During production, 500kg allocated:
    - Quantity now available: 300kg.
5. Manager views inventory:
    - Sees batch code, vendor, status, available qty.

Module 3: Production Process Logging

---

1. High-Level Design (HLD)

Objective:

Enable shift supervisors to log every stage of the cellulose production process in a structured, auditable, and traceable manner—ensuring material usage, machine operation, timings, and outputs are systematically recorded.

Core Production Stages:

1. Pre-treatment
2. Drying
3. Grinding
4. Sieving
5. Blending

Key Features:

- Stage-wise process data entry
- Time in/out, input-output quantities
- Machine ID, operator ID
- Material batch linkage
- Inter-stage traceability
- Automatic stage locking after shift
- Supervisor-only editable entries

- Production logs feed into dashboard & anomaly engine

---

2. Low-Level Design (LLD)

2.1. Database Models

```
class ProductionBatch(models.Model):
    batch_code = models.CharField(max_length=50, unique=True)
    date = models.DateField()
    shift = models.CharField(max_length=2, choices=[("A", "Shift A"), ("B", "Shift B"), ("C", "Shift C")])
    created_by = models.ForeignKey(User, on_delete=models.SET_NULL, null=True)

class ProcessStage(models.Model):
    name = models.CharField(max_length=50)  # e.g., Pre-treatment, Drying, etc.

class StageLog(models.Model):
    production_batch = models.ForeignKey(ProductionBatch, on_delete=models.CASCADE)
    stage = models.ForeignKey(ProcessStage, on_delete=models.CASCADE)
    input_material_batch = models.ForeignKey(MaterialBatch, on_delete=models.SET_NULL, null=True)
    input_quantity = models.DecimalField(max_digits=10, decimal_places=2)
    output_quantity = models.DecimalField(max_digits=10, decimal_places=2)
    machine_id = models.CharField(max_length=50)
    operator_name = models.CharField(max_length=100)
    start_time = models.DateTimeField()
    end_time = models.DateTimeField()
    remarks = models.TextField(null=True, blank=True)
```

2.2. API Endpoints (Django DRF)

Endpoint	Method	Role	Description
/api/production-batch/	GET/POST	Supervisor	Create/view production batch
/api/stage-log/	POST	Supervisor	Log stage-wise details
/api/stage-log/<id>/	PUT	Supervisor	Update (only same shift-day)
/api/stage-log/batch/<batch_code>/	GET	All roles	View full batch flow

---

2.3. Business Rules

- Each stage can be logged once per production batch
- Start time < End time is enforced
- Input quantity cannot be less than output (flag for review)
- Once shift is over, logs are locked
- Manual changes must be approved by Admin
- Total material usage must match raw material allocations

2.4. UI Components (React)

Page	Components
Production Entry	ProductionForm, StageSelector, MaterialBatchDropdown
Stage Logging	StageLogForm, TimePicker, MachineIDInput
Batch Flow Viewer	TimelineView, StageStatusCard
Supervisor Logbook	ShiftFilter, EditModal, RemarksPanel

2.5. Validation Logic

```
# in serializers.py
def validate(self, data):
    if data['start_time'] >= data['end_time']:
```

```
        raise serializers.ValidationError("End time must be after start time.")
    if data['input_quantity'] < data['output_quantity']:
        raise serializers.ValidationError("Output cannot exceed input.")
    return data
```

2.6. Audit + Locking Mechanism

- A background task (Celery) runs every hour to:
    - Lock shift data post shift time (configurable per plant)
    - Mark incomplete stage logs
- All updates beyond shift require Manager override

2.7. Sample Use Case

1. Supervisor creates Batch MCC-24-051-A for Shift A.
2. Logs Pre-treatment:
    - Input: 800kg, Output: 770kg, Machine: MCH-101, Time: 6:10AM–7:40AM
3. Logs Drying:
    - Input: 770kg, Output: 760kg, Machine: MCH-202
4. Logs all stages with time, operator, and remarks.
5. At 2:00 PM (Shift A ends), entries are locked.
6. Manager reviews logs next day from dashboard.

Module 4: Quality Control (QC) and Testing Management

---

1. High-Level Design (HLD)

Objective:

Capture, manage, and track QC parameters for raw materials, in-process materials, and final products with role-based approvals, test thresholds, and deviation detection.

QC Points:

- Incoming Material QC (linked to MaterialBatch)
- In-Process QC (stage-level checks during production)

- Final Product QC (post last production stage)

Key Features:

- Define test parameters per material type
- Perform test data entry by QA role
- Threshold-based anomaly detection
- QC status: PENDING / PASS / FAIL / HOLD
- Attach lab report PDFs/images
- Override flow with Manager approval
- Automatic traceability to batch and process stage

---

2. Low-Level Design (LLD)

2.1. Database Models

```
class QCParameter(models.Model):
    material = models.ForeignKey(RawMaterial, on_delete=models.CASCADE)
    name = models.CharField(max_length=50)  # e.g., Moisture %, pH
    min_value = models.FloatField(null=True, blank=True)
    max_value = models.FloatField(null=True, blank=True)
    unit = models.CharField(max_length=20)

class QCTest(models.Model):
    source_type = models.CharField(max_length=20, choices=[('RAW', 'Raw Material'), ('PROCESS', 'In-Process'), ('FINAL', 'Final Product')])
    source_id = models.CharField(max_length=50)  # Can be MaterialBatch ID, StageLog ID, or ProductionBatch ID
    performed_by = models.ForeignKey(User, on_delete=models.SET_NULL, null=True)
    date = models.DateTimeField(auto_now_add=True)
```

```
    status = models.CharField(max_length=10, choices=[('PENDING', 'Pending'), ('PASS', 'Pass'), ('FAIL', 'Fail'), ('HOLD', 'Hold')])
    remarks = models.TextField(null=True, blank=True)
    attachment = models.FileField(upload_to='qc_reports/', null=True, blank=True)


class QCTestResult(models.Model):
    test = models.ForeignKey(QCTest, related_name='results', on_delete=models.CASCADE)
    parameter = models.ForeignKey(QCParameter, on_delete=models.CASCADE)
    observed_value = models.FloatField()
    is_within_spec = models.BooleanField()
```

2.2. API Endpoints (Django DRF)

Endpoint	Method	Access	Description
/api/qc/parameters/	GET/POST	Admin	Define test parameters
/api/qc/tests/	POST	QA	Log QC test
/api/qc/tests/<id>/	GET/PUT	QA	View/update result
/api/qc/final-status/<source_id>/	GET	All Roles	Fetch overall QC decision
/api/qc/override/	POST	Manager	Manager overrides test result (requires comment)

2.3. Business Rules

- All QC entries must be timestamped and assigned to a responsible QA user.
- Observed values outside min/max range → auto-flag is_within_spec = False.
- If one or more parameters fail → QC status = HOLD unless overridden.
- A batch must not proceed to next stage without QC pass (unless overridden).
- Test data older than 24 hours = locked from editing.

2.4. UI Components (React)

Page	Components
QC Dashboard	FilterBar, QCTestTable, StatusChip
QC Entry Form	ParameterRepeater, RangeValidator, UploadLabReport
QC Status Viewer	TestResultTable, OverrideDialog, CommentsTimeline
Attachment Preview	PDFViewer, ImageZoomer

2.5. QC Evaluation Logic

```
def evaluate_qc(test_id):
    test = QCTest.objects.get(id=test_id)
    failed_params = test.results.filter(is_within_spec=False).count()
    if failed_params == 0:
        test.status = "PASS"
    else:
        test.status = "HOLD"
    test.save()
```

2.6. Validation & Access Control

- Only QA can create/update test entries
- Manager required to override failed result
- Reports must be PDF/JPG/PNG ≤ 10MB
- Comments + attachments required for any override

2.7. Sample Use Case

1. QA performs final QC on batch MCC-24-051-A:
    - Moisture % = 5.2 (within 4–6) → PASS
    - Particle size = 65µm (spec 40–60) → FAIL

2. System auto-sets status = HOLD.
3. Manager reviews → determines acceptable variance → overrides to PASS with remark.
4. Report attached. Status updated.
5. Entry locked after 24h. Dashboard updated.

Module 5: Inventory & Material Movement Management

---

1. High-Level Design (HLD)

Objective:

Digitally track raw material, in-process material, and finished goods movement across storage areas, production floor, and dispatch—ensuring real-time visibility, stock consistency, and traceability.

Scope Includes:

- Raw Material inward tracking
- Internal transfers (store → production → QC → store)
- Finished product packaging and dispatch logging
- Stock adjustment with role-based audit
- Integration with Production & QC modules
- Alerts for low/reorder level stocks

Key Features:

- GRN (Goods Receipt Note) based inward logging
- Bin/Location-level stock tracking
- FIFO-based material consumption logic
- Reconciliation & audit logging
- Material status: AVAILABLE / RESERVED / CONSUMED / REJECTED
- Dashboard: Current stock, movement history, anomalies

---

2. Low-Level Design (LLD)

2.1. Database Models

```
class InventoryItem(models.Model):
    material = models.ForeignKey(RawMaterial, on_delete=models.CASCADE)
    batch_code = models.CharField(max_length=50)
    quantity = models.DecimalField(max_digits=10, decimal_places=2)
    uom = models.CharField(max_length=20)  # e.g., KG, L
    location = models.CharField(max_length=100)  # Store A, Line B, etc.
    status = models.CharField(max_length=20, choices=[('AVAILABLE', 'Available'),
('RESERVED', 'Reserved'), ('CONSUMED', 'Consumed'), ('REJECTED', 'Rejected')])
    expiry_date = models.DateField(null=True, blank=True)
    created_at = models.DateTimeField(auto_now_add=True)

class StockMovement(models.Model):
    item = models.ForeignKey(InventoryItem, on_delete=models.CASCADE)
    from_location = models.CharField(max_length=100)
    to_location = models.CharField(max_length=100)
    moved_by = models.ForeignKey(User, on_delete=models.SET_NULL, null=True)
    timestamp = models.DateTimeField(auto_now_add=True)
    quantity_moved = models.DecimalField(max_digits=10, decimal_places=2)
    purpose = models.CharField(max_length=100)  # e.g., For Production, Return to Store,
Sample Transfer
```

2.2. API Endpoints (Django DRF)

Endpoint	Method	Role	Description
/api/inventory/inward/	POST	Storekeeper	Log new material into stock
/api/inventory/transfer/	POST	Storekeeper/Supervisor	Move material from one location to another
/api/inventory/<batch_code>/	GET	All Roles	Get batch-level stock detail
/api/inventory/consumption/	POST	System	Auto-update on production stage
/api/inventory/adjust/	POST	Manager	Manual stock adjustment with reason

---

2.3. Business Rules

- All GRNs must reference Purchase Orders
- No transfer allowed of rejected or expired batches
- FIFO enforced for raw material drawdowns
- QC “HOLD” status blocks movement
- Production stages consume stock automatically
- Reconciliation reports weekly/monthly

2.4. UI Components (React)

Page	Components
Inventory Dashboard	SummaryCard, LowStockWarning, DateRangeSelector
GRN Entry	GRNForm, POAutoFetch, FileUpload
Stock Transfer	TransferForm, BatchDropdown, LocationSelector
Movement History	MovementTable, FilterBar, CSVExportButton
Adjustment Panel	AuditModal, ReasonSelector

2.5. FIFO & Consumption Logic

```
def get_next_available_stock(material_id, required_qty):
    available_items = InventoryItem.objects.filter(
        material_id=material_id,
        status='AVAILABLE',
    ).order_by('created_at')

    selected_items = []
    qty_accumulated = 0
    for item in available_items:
        if qty_accumulated >= required_qty:
            break
        qty = min(item.quantity, required_qty - qty_accumulated)
        selected_items.append((item, qty))
        qty_accumulated += qty
    return selected_items
```

2.6. Validation & Access Control

- Only Storekeeper can create inward/transfer entries
- Supervisor can view and request transfers
- Manager required for manual adjustments
- Quantity can't exceed available balance
- All stock actions are logged and timestamped

2.7. Sample Use Case

1. Storekeeper receives 500 kg of MCC Raw.
    - Creates GRN → Enters batch code, expiry, QC status: PENDING
2. QA clears QC → Status: AVAILABLE
3. Supervisor initiates transfer to Production Floor (Shift A)
    - FIFO batch picked automatically
4. During production logging, 300 kg consumed → Status updated: CONSUMED
5. Leftover 200 kg moved back to store
6. End of week → Manager audits mismatch → posts an adjustment with reason: "spillage"

Module 6: Anomaly Detection & Monitoring System

---

1. High-Level Design (HLD)

Objective:

Detect human errors, operational inconsistencies, or data anomalies across the plant's manually entered and system-generated records, triggering alerts and generating patterns to improve reliability and auditability.

Anomaly Types Covered:

- Out-of-range input data (supervisor entry errors)
- Missed entries or timing mismatches (shift-wise delays)
- Logical sequence violations (e.g., production before QC pass)
- Repetitive entry patterns from specific users (copy-paste abuse)
- QC value deviation trend (outliers or drift)
- Stock mismatch (paper vs. system vs. consumption)

Key Features:

- Real-time validation feedback on data entry
- Scheduled anomaly scans (hourly/nightly)
- Alert dashboard with severity levels

- Configurable rule engine
- Notification system (email, SMS, in-app)
- Anomaly approval/resolution workflow

---

2. Low-Level Design (LLD)

2.1. Rule Engine Design

A rule is defined as:

json

```
{
  "name": "QC Value Out of Range",
  "source": "QCTestResult",
  "field": "observed_value",
  "condition": "outside_min_max",
  "severity": "high",
  "auto_flag": true
}
```

Rule Model:

```
class AnomalyRule(models.Model):
    name = models.CharField(max_length=100)
    source_table = models.CharField(max_length=100)  # e.g., QCTestResult
    field_name = models.CharField(max_length=100)  # e.g., observed_value
    condition = models.CharField(max_length=50)  # 'outside_min_max', 'missing', 'duplicated_entry'
    severity = models.CharField(max_length=20, choices=[('low', 'Low'), ('medium', 'Medium'), ('high', 'High')])
    is_active = models.BooleanField(default=True)
```

```
    notify_roles = ArrayField(models.CharField(max_length=50))  # e.g., ['Manager', 'QA']
```

Anomaly Model:

```
class DetectedAnomaly(models.Model):
    rule = models.ForeignKey(AnomalyRule, on_delete=models.SET_NULL, null=True)
    source_id = models.CharField(max_length=100)  # ID of related record
    detected_at = models.DateTimeField(auto_now_add=True)
    status = models.CharField(max_length=20, choices=[('open', 'Open'), ('resolved', 'Resolved'), ('ignored', 'Ignored')])
    severity = models.CharField(max_length=20)
    description = models.TextField()
    resolved_by = models.ForeignKey(User, null=True, blank=True, on_delete=models.SET_NULL)
    resolved_at = models.DateTimeField(null=True, blank=True)
```

2.2. Anomaly Detection Service Layer

- Inline Validator: Live checks during data entry (form validation + backend)
- Batch Processor (Celery Task): Scans all entries every hour/night
- Alert Dispatcher: Sends anomaly notifications based on severity + role
- Resolution Tracker: Maintains audit log of anomaly lifecycle

---

2.3. API Endpoints (DRF)

Endpoint	Method	Role	Description
/api/anomalies/rules/	GET/POST	Admin	Create or update detection rules
/api/anomalies/	GET	All	View detected anomalies
/api/anomalies/<id>/resolve/	POST	Manager/QA	Resolve anomaly with comment
/api/anomalies/<id>/ignore/	POST	Manager	Ignore a false positive

---

2.4. UI Components (React)

Page	Components
Anomaly Dashboard	SeverityFilter, AnomalyTable, ResolutionProgress
Anomaly Detail View	MetadataDisplay, RelatedRecordLinker, ResolveForm
Rule Editor (Admin)	RuleForm, FieldSelector, ConditionSelector

---

2.5. Sample Detection Logic

QC Value Out-of-Range

```
for result in QCTestResult.objects.select_related('parameter'):
    if result.observed_value < result.parameter.min_value or result.observed_value > result.parameter.max_value:
        create_anomaly(
            rule="QC Value Out of Range",
            source_id=result.id,
            description=f"{result.parameter.name} value {result.observed_value} out of range"
        )
```

Missing Shift Log

```
expected_shifts = get_all_shifts_for_day(today)
for shift in expected_shifts:
    if not ShiftLog.objects.filter(shift_id=shift.id).exists():
        create_anomaly(rule="Missing Shift Entry", source_id=shift.id, ...)
```

2.6. Validation & Security

- Only designated roles (Manager, QA) can resolve/ignore anomalies
- All actions logged with timestamp and user ID
- Rule changes allowed only by Admin

- Notifications logged for audit

2.7. Sample Use Case

1. Supervisor enters production data → skips moisture input.
    - System raises "Missing Parameter" anomaly → Level: Medium.
2. QC uploads test with particle size = 89 µm (spec: 40–60)
    - Detected anomaly: "Out of Range QC"
    - System notifies QA & Manager
3. Manager reviews → Marks as resolved with override explanation.
4. Rule engine records override frequency and flags if abuse is suspected.

Module 7: Dashboard & Reporting System

---

1. High-Level Design (HLD)

Objective:

Provide a real-time, role-based visual overview of the plant's operations, performance, inventory, production efficiency, and anomaly trends—accessible to Managers, Supervisors, QA, and Admin.

Scope:

- Real-time dashboards for production, inventory, QC, and anomaly trends
- Custom report generation (daily, weekly, monthly)
- Export support (CSV, PDF)
- KPI indicators (shift efficiency, rejection %, stock balance)
- Role-based data visibility
- Drill-down capability for reports

---

2. Low-Level Design (LLD)

2.1. Dashboard Categories

a. Production Dashboard

- Total batches produced today/week/month
- Shift-wise throughput
- Stage-wise process timing
- Delay indicators

b. QC Dashboard

- Pass/fail statistics
- Common rejection reasons
- Parameter-wise QC failure trends

c. Inventory Dashboard

- Stock levels by material/category/location
- Fast-moving vs. slow-moving items
- Reorder alert status

d. Anomaly Dashboard

- Open/resolved anomalies by category
- Rule violation heatmap
- Response time SLAs

e. Performance KPIs

- Production vs. Plan %
- Downtime %
- QC rejection %
- Average resolution time (anomalies)

2.2. DB Views / Aggregation Models

Use PostgreSQL materialized views or background Celery tasks for pre-aggregated data.

sql

```
CREATE MATERIALIZED VIEW prod_kpi_summary AS
SELECT
```

```
  DATE(production_date) AS date,
  shift,
  COUNT(*) AS total_batches,
  SUM(CASE WHEN status = 'Completed' THEN 1 ELSE 0 END) AS completed_batches,
  AVG(duration) AS avg_batch_time
FROM production_batch
GROUP BY DATE(production_date), shift;
```

Or via Django ORM using aggregation in a scheduled task:

```
Batch.objects.filter(...)
   .annotate(date=TruncDay('created_at'))
   .values('date', 'shift')
   .annotate(total=Count('id'), avg_duration=Avg('duration'))
```

2.3. API Endpoints (DRF)

Endpoint	Method	Role	Description
/api/dashboard/production/	GET	All	Summary of production metrics
/api/dashboard/qc/	GET	QA, Manager	QC pass/fail and deviation chart
/api/dashboard/inventory/	GET	Storekeeper, Manager	Stock stats and movement trends
/api/dashboard/anomalies/	GET	Manager	Open issues, violation trends
/api/reports/custom/	POST	Manager	Generate and download CSV/PDF

2.4. Frontend Components (React + Chart.js or Recharts)

Page	Components
Dashboard Home	KPICard, DateRangeSelector, RoleTabs

Page	Components
Production	LineChart (output over time), ShiftTable
QC	StackedBarChart (pass/fail), PieChart (failure reason %)
Inventory	StockLevelGauge, HeatMap (stock movement)
Anomalies	TimelineView, AlertSeverityPie
Reports	ReportFilterForm, DownloadButton

2.5. Sample Report Config

json

```
{
  "report_name": "Weekly QC Summary",
  "filters": {
    "start_date": "2025-05-01",
    "end_date": "2025-05-07",
    "product_line": "Microcrystalline Cellulose"
  },
  "format": "PDF",
  "include_graphs": true,
  "scheduled": false
}
```

2.6. Access Control

- Manager: Full access to all dashboards + report generation
- Supervisor: Limited access to production/inventory data for their shift
- QA: QC metrics and anomaly summary
- Admin: Configures KPI formulas and data pipelines

2.7. Notifications & Scheduling

- Option to auto-email reports daily/weekly to stakeholders
- Report generation tasks offloaded to Celery workers
- Auto-deletion or archival of outdated reports to S3 or GDrive

---

2.8. Use Case Example

1. Morning Routine: Manager logs in → sees today's shift production, previous night QC failures, and open anomalies.
2. At 9 PM: System auto-generates the daily report → emails PDF to all department heads.
3. Inventory Warning: Reorder level alert pops up for "Binder - Type C" → visible in stock dashboard.

Module 8: Admin & Configuration Management

---

1. High-Level Design (HLD)

Objective:

Provide centralized control to manage master data, users, roles, shift schedules, rule engines, parameter thresholds, and system behavior configurations. Enables business logic agility without code-level changes.

Responsibilities:

- User & role management
- Shift and plant configuration
- Master data management (products, materials, machines)
- QC parameter configuration
- Rule engine editor (for anomaly detection)
- Access control (RBAC)
- System preferences (e.g., notification modes, report templates)

---

2. Low-Level Design (LLD)

2.1. Core Models

- User & Role Model

```
class Role(models.Model):
    name = models.CharField(max_length=50)  # Admin, Supervisor, QA, Manager

class User(AbstractUser):
    role = models.ForeignKey(Role, on_delete=models.PROTECT)
    phone = models.CharField(max_length=15)
    is_active = models.BooleanField(default=True)
```

- Shift & Plant Configuration

```
class Shift(models.Model):
    name = models.CharField(max_length=50)  # Shift A, B, C
    start_time = models.TimeField()
    end_time = models.TimeField()
    supervisor = models.ForeignKey(User, on_delete=models.SET_NULL, null=True, blank=True)

class Plant(models.Model):
    name = models.CharField(max_length=100)
    location = models.CharField(max_length=100)
```

- Master Data Entities

```
class Product(models.Model):
    name = models.CharField(max_length=100)
    code = models.CharField(max_length=20, unique=True)
    description = models.TextField()

class Material(models.Model):
    name = models.CharField(max_length=100)
    code = models.CharField(max_length=20, unique=True)
    uom = models.CharField(max_length=10)  # kg, L

class Machine(models.Model):
    name = models.CharField(max_length=100)
    serial_number = models.CharField(max_length=50, unique=True)
    location = models.CharField(max_length=100)
```

2.2. QC Parameter Config

```
class QCParameter(models.Model):
    name = models.CharField(max_length=100)
    product = models.ForeignKey(Product, on_delete=models.CASCADE)
    min_value = models.DecimalField(max_digits=6, decimal_places=2)
    max_value = models.DecimalField(max_digits=6, decimal_places=2)
    unit = models.CharField(max_length=20)
```

2.3. Rule Engine Admin

Rules for Module 6 can be modified via GUI:

```
class AnomalyRule(models.Model):
    name = models.CharField(max_length=100)
    condition = models.TextField()  # Optional DSL or JSON expression
    severity = models.CharField(max_length=10)
    active = models.BooleanField(default=True)
```

2.4. Access Control System (RBAC)

Permission Matrix Example:

Action	Supervisor	QA	Manager	Admin
Create QC Rule	❌	✅	✅	✅
View Inventory	✅	❌	✅	✅
Assign Shift	❌	❌	✅	✅
Add User	❌	❌	❌	✅

Implemented using Django's permissions, groups, and middleware-based authorization hooks.

2.5. System Preferences

```
class SystemPreference(models.Model):
    key = models.CharField(max_length=100, unique=True)
    value = models.TextField()
    description = models.TextField(blank=True)
```

Examples:

- "report.auto_send": "true"
- "notification.channel": "email,sms"
- "default_qc_range_tolerance": "±5%"

2.6. API Endpoints (DRF)

Endpoint	Method	Role	Description
/api/admin/users/	GET/POST	Admin	Create, update users
/api/admin/shifts/	GET/POST	Admin, Manager	Configure shifts
/api/admin/qc-params/	GET/POST	QA, Admin	Manage QC rules
/api/admin/rules/	GET/POST	Admin	Modify anomaly rules
/api/admin/preferences/	GET/POST	Admin	Set system behaviors

2.7. UI Components (React)

Section	Components
User Management	UserListTable, UserFormModal, RoleFilter
Master Data	ProductManager, MaterialTable, MachineConfigForm
Shift Scheduler	ShiftCalendarView, ShiftAssignmentModal
Preferences	KeyValueEditor, SaveButton
Rule Engine	RuleList, RuleEditor, JSONValidator

2.8. Workflow Example

1. Admin logs in → Adds new product: MCC Type R-102
2. QA adds QC parameters: Moisture (2–5%), pH (6.5–7.5)
3. Manager creates Shift D from 10 PM to 6 AM
4. Admin defines a rule: “Reject if moisture > 5%” with severity = HIGH
5. System now triggers alerts during QC entry using this config

---

Security & Audit

- Every config change is logged (AdminActionLog)
- Deletion is soft-delete (for traceability)
- Role-restricted access using JWT claims

Module 9: Notification & Escalation System

---

1. High-Level Design (HLD)

Objective:

Ensure critical events like anomalies, delays, QC failures, inventory shortages, or supervisor input errors are notified to the right stakeholders, with a tiered escalation mechanism to guarantee accountability and timely resolution.

Key Capabilities:

- Real-time notifications via Email, SMS, Push (web/mobile)
- Configurable escalation levels based on severity and response SLAs
- Event-driven integration (QC failure, material understock, unclosed shift, etc.)
- Daily/weekly digest reports for non-critical events

---

2. Low-Level Design (LLD)

2.1. Notification Types

Event Type	Trigger	Target
Anomaly Detected	Rule match	Supervisor, Manager
QC Failure	Out-of-spec value	QA, Manager
Inventory Below Threshold	Reorder level crossed	Storekeeper, Procurement
Shift Not Closed	No entry beyond shift end	Supervisor, Admin
Input Errors	Duplicate/missing/suspicious entries	Supervisor, Admin
Escalation Triggered	SLA not met	Next-level Manager

2.2. Models

```
class Notification(models.Model):
    recipient = models.ForeignKey(User, on_delete=models.CASCADE)
```

```
    message = models.TextField()
    event_type = models.CharField(max_length=50)
    related_object_id = models.IntegerField(null=True, blank=True)
    seen = models.BooleanField(default=False)
    created_at = models.DateTimeField(auto_now_add=True)

class EscalationRule(models.Model):
    event_type = models.CharField(max_length=50)
    trigger_after_minutes = models.IntegerField()
    escalate_to = models.ForeignKey(User, on_delete=models.PROTECT)
    active = models.BooleanField(default=True)
```

2.3. Notification Channels

a. Email

- SMTP server (e.g., Amazon SES, Mailgun)
- HTML templated emails for critical alerts

b. SMS

- Twilio or local SMS Gateway
- Limited to brief, urgent events

c. In-App / Web Push

- Real-time with Django Channels + WebSocket
- Seen/unseen tracking

d. Mobile Push (optional)

- Firebase Cloud Messaging (FCM)
- Used for PWA or native app variants

2.4. Notification Engine (Service Layer)

```
def send_notification(event_type, message, recipients, object_id=None):
    for user in recipients:
        Notification.objects.create(
            recipient=user,
            event_type=event_type,
            message=message,
            related_object_id=object_id
        )
        if user.prefers_email:
            send_email(user.email, subject=event_type, body=message)
        if user.prefers_sms:
            send_sms(user.phone, message)
        if user.prefers_push:
            send_websocket_push(user.id, message)
```

2.5. Escalation Workflow

```
@shared_task
def check_escalations():
    for rule in EscalationRule.objects.filter(active=True):
        events = Notification.objects.filter(
            event_type=rule.event_type,
            seen=False,
            created_at__lt=timezone.now() - timedelta(minutes=rule.trigger_after_minutes)
        )
```

```
    for evt in events:
        send_notification(
            event_type=f"{evt.event_type} (Escalated)",
            message=f"Escalated: {evt.message}",
            recipients=[rule.escalate_to],
            object_id=evt.related_object_id
        )
```

Scheduled every 10 minutes using Celery.

2.6. API Endpoints (DRF)

Endpoint	Method	Description
/api/notifications/	GET	List of notifications for current user
/api/notifications/mark-seen/	POST	Mark notifications as read
/api/admin/escalation-rules/	GET/POST	Configure escalation timings and targets

---

2.7. Frontend Components (React)

Component	Description
NotificationBell	Icon with real-time count
NotificationList	Dropdown or modal with latest events
EscalationConfigForm	Admin UI to manage rules
NotificationToast	Pop-up alerts for real-time delivery

2.8. Sample Notification Scenarios

- QC Violation:
    - Supervisor A enters pH = 8.2 for a batch where max = 7.5
    - Notification to QA and Manager
    - If not marked seen or resolved in 30 minutes, escalated to Plant Head

- Stock Alert:
    - “Binder X” stock falls below 10kg
    - Alert to Storekeeper
    - Escalated to Procurement Head in 1 hour if no action

2.9. Notification Preferences Model (Optional)

```
class UserNotificationPreference(models.Model):
    user = models.OneToOneField(User, on_delete=models.CASCADE)
    prefers_email = models.BooleanField(default=True)
    prefers_sms = models.BooleanField(default=False)
    prefers_push = models.BooleanField(default=True)
```

Audit Trail

- Every notification is logged
- Delivery status stored for SMS/Email
- Escalation history tracked for compliance

Module 10: Reporting & Analytics

---

1. High-Level Design (HLD)

Objective:

Deliver actionable insights and aggregated data views to managers and admins across production, quality, inventory, and anomaly trends. Enable report generation, KPI tracking, and decision-making dashboards.

🔧 Key Functionalities:

- Standard reports (Shift Summary, Batch QC, Material Usage, etc.)
- Ad-hoc/custom report generator
- Charts/Graphs for anomaly trends, plant performance

- Export options (PDF, Excel)
- Dashboard with filterable KPIs
- Scheduled report delivery via Email

---

2. Low-Level Design (LLD)

2.1 Core Reports

Report Name	Frequency	Filters	Format
Shift Production Summary	Per shift	Date, Plant, Shift	Table, PDF
QC Compliance Report	Daily/Weekly	Date, Product, Plant	Table, Excel
Material Consumption Report	Monthly	Material, Date	Chart, Table
Anomaly Trend Analysis	Weekly	Severity, Rule	Line/Bar Graph
Inventory Usage vs Threshold	Monthly	Material, UOM	Excel, Chart
Operator Input Accuracy	Monthly	Supervisor, Field	Table, Pie

2.2 Models & Schemas

Report Configuration

```
class ReportConfig(models.Model):
    name = models.CharField(max_length=100)
    report_type = models.CharField(max_length=50)  # shift_summary, qc_compliance
    parameters = JSONField()  # { "plant_id": 1, "date": "2024-05-01" }
    schedule_cron = models.CharField(max_length=20, null=True, blank=True)
    recipient = models.ForeignKey(User, on_delete=models.CASCADE)
    format = models.CharField(max_length=10, choices=[("pdf", "PDF"), ("xls", "Excel")])
```

2.3 Dashboard Components

KPI	Data Source	Formula
Avg QC Pass %	QC records	passed / total × 100
Material Shortage Events	Inventory logs	count(status="low")
Shift Delay Rate	Shift logs	delayed / total shifts
Operator Accuracy	Manual input logs	(valid / total) × 100
Top 5 Frequent Anomalies	Anomaly logs	group by rule_id

2.4 Report Generation Pipeline

Step-by-step:

1. User requests or schedules report
2. DataService pulls filtered data (batch-wise/shift-wise/etc.)
3. TransformService cleans, joins, and computes derived metrics
4. RenderService exports to PDF/XLS using templates
5. NotifyService emails the report (if scheduled)

2.5 Key Services

```
class ReportBuilder:
    def __init__(self, report_type, filters):
        self.type = report_type
        self.filters = filters

    def generate(self):
        raw_data = DataService.fetch(self.type, self.filters)
        transformed = TransformService.clean_and_group(raw_data)
        file_path = RenderService.render_to_pdf_or_excel(transformed, self.type)
        return file_path
```

2.6 APIs (DRF)

Endpoint	Method	Role	Description
/api/reports/available/	GET	All	Lists available report types
/api/reports/generate/	POST	Admin/Manager	Generates on-demand report
/api/reports/schedule/	POST	Admin	Schedules recurring report
/api/reports/history/	GET	All	Report download logs/history

2.7 Frontend Components (React)

Component	Purpose
ReportGeneratorForm	Select report, filters, and format
KPIWidgets	Visual tiles with real-time KPI cards
TrendCharts	Bar, Line, Pie charts for anomalies, shifts, etc.
SchedulerPanel	Manage auto-report schedules
ReportHistoryTable	Download past reports

2.8 Scheduled Reports (Celery + Cron)

- report_scheduler.py checks ReportConfig entries hourly
- If cron matches current time → triggers generate() and emails result

---

2.9 Export Engine

- PDF: WeasyPrint with HTML+CSS templates
- Excel: openpyxl or pandas.DataFrame.to_excel()
- Optionally, add watermarking & version stamp

END OF DOCUMENT.