MediTrack - Comprehensive System Requirements

Project Overview

System Name: MediTrack - Medicine Stock and Distribution System
Tech Stack: ReactJS + Spring Boot + MySQL + Flask (AI Module)
Target Users: Admin, Suppliers, Pharmacy/Distributors
Location Context: Nepal Healthcare System

---

1. FRONTEND REQUIREMENTS

1.1 Public Pages (No Authentication Required)

Page 1: Landing Page (`/`)

Purpose: Introduction to MediTrack system
Components:

- Hero section with system overview
- Key features showcase (Real-time tracking, Expiry alerts, AI forecasting)
- Statistics dashboard (Total medicines tracked, Active suppliers, Hospitals served)
- Call-to-action buttons (Login, Register, Learn More)
- Footer with contact information

Page 2: Login Page (`/login`)

Purpose: User authentication
Components:

- Email/Username input field
- Password input field (with show/hide toggle)
- "Remember Me" checkbox
- Login button
- "Forgot Password?" link
- "Register New Account" link
- Role selection dropdown (optional: pre-filled based on registration)

Page 3: Registration Page (`/register`)

Purpose: New user account creation
Components:

- Full Name input
- Email input
- Phone Number input
- Password input (with strength indicator)
- Confirm Password input
- Role selection (Supplier/Pharmacy)
- Organization Name input
- License Number input (for verification)
- Address input
- Terms & Conditions checkbox
- Register button

- "Already have an account? Login" link

Page 4: Forgot Password Page (`/forgot-password`)

Purpose: Password recovery
Components:

- Email input field
- Send Reset Link button
- Back to Login link
- Success message display

1.2 Admin Dashboard Pages (Role: ADMIN)

Page 5: Admin Dashboard Home (`/admin/dashboard`)

Purpose: Overview of entire system
Components:

- Total Medicines Count card
- Total Suppliers Count card
- Total Pharmacies Count card
- Low Stock Medicines Count (alert badge)
- Near Expiry Medicines Count (alert badge)
- Recent Activity Timeline
- Stock Level Chart (Chart.js - Bar/Line)
- Distribution Trends Chart (Chart.js - Area)
- Top 10 Most Used Medicines table
- Quick Action buttons (Add Medicine, View Reports, Manage Users)

Page 6: Medicine Management (`/admin/medicines`)

Purpose: Complete medicine inventory management
Components:

- Search bar (by name, category, batch number)
- Filter options (Category, Status, Supplier)
- "Add New Medicine" button
- Medicines Table with columns:
  - Medicine Name
  - Category
  - Generic Name
  - Manufacturer
  - Current Stock
  - Unit Price
  - Status (Available/Low Stock/Out of Stock)
  - Actions (View, Edit, Delete)
- Pagination controls
- Export button (CSV/PDF)
- Bulk Upload button

Page 7: Add/Edit Medicine Modal (`/admin/medicines/add` or `/admin/medicines/edit/:id`)

Purpose: Add or modify medicine details
Components:

- Medicine Name input (required)

- Generic Name input
- Category dropdown (Antibiotics, Painkillers, Vitamins, etc.)
- Manufacturer input
- Description textarea
- Unit Price input
- Reorder Level input (for low stock alert)
- Supplier dropdown (multi-select)
- Medicine Image upload
- Save button
- Cancel button

Page 8: Batch Management (`/admin/batches`)

Purpose: Track medicine batches with expiry dates
Components:

- Search bar (by batch number, medicine name)
- Filter options (Expiry Status, Supplier, Date Range)
- "Add New Batch" button
- Batches Table with columns:
  - Batch Number
  - Medicine Name
  - Quantity
  - Manufacturing Date
  - Expiry Date
  - Days to Expiry (color-coded: red < 30 days, yellow < 90 days)
  - Supplier Name
  - Status (Active/Expired/Near Expiry)
  - Actions (View, Edit, Delete)
- Expiry Alert Banner (showing critical batches)
- Export button

Page 9: Add/Edit Batch Modal

Purpose: Record new batch or update existing
Components:

- Medicine dropdown (searchable)
- Batch Number input
- Quantity input
- Manufacturing Date picker
- Expiry Date picker (with auto-calculation of shelf life)
- Supplier dropdown
- Purchase Price input
- Storage Location input
- Notes textarea
- Save button
- Cancel button

Page 10: Stock Management (`/admin/stock`)

Purpose: Monitor current stock levels across locations
Components:

- Search and filter options
- Location filter (Warehouse, Pharmacy branches)
- Stock Level Chart (visual representation)
- Stock Table with columns:

    - Medicine Name
    - Location
    - Current Quantity
    - Reorder Level
    - Status indicator
    - Last Updated
    - Actions (Adjust Stock, Transfer)
- Low Stock Alert section
- "Stock Adjustment" button
- "Stock Transfer" button

Page 11: Stock Adjustment Modal

Purpose: Add or remove stock quantities
Components:

- Medicine dropdown
- Current Stock display (read-only)
- Adjustment Type (Add/Remove)
- Quantity input
- Reason dropdown (Received from supplier, Damaged, Expired, etc.)
- Batch Number input
- Date picker
- Notes textarea
- Submit button

Page 12: Supplier Management (`/admin/suppliers`)

Purpose: Manage supplier information
Components:

- "Add New Supplier" button
- Search bar
- Suppliers Table with columns:
    - Supplier Name
    - Contact Person
    - Email
    - Phone
    - Address
    - Total Supplies
    - Status (Active/Inactive)
    - Rating (star-based)
    - Actions (View, Edit, Delete, View History)
- Pagination

Page 13: Add/Edit Supplier Modal

Purpose: Supplier registration and updates
Components:

- Company Name input
- Contact Person Name input
- Email input
- Phone Number input
- Address textarea
- License Number input
- Medicines Supplied (multi-select dropdown)
- Payment Terms dropdown

- Status toggle (Active/Inactive)
- Notes textarea
- Save button

Page 14: Pharmacy/Distributor Management (`/admin/pharmacies`)

Purpose: Manage pharmacy and distributor accounts
Components:

- "Add New Pharmacy" button
- Search and filter options
- Pharmacies Table with columns:
  - Pharmacy Name
  - Owner/Manager Name
  - Email
  - Phone
  - Location
  - License Number
  - Status (Active/Inactive)
  - Registration Date
  - Actions (View, Edit, Delete, View Orders)

Page 15: Distribution Management (`/admin/distribution`)

Purpose: Track medicine distribution to pharmacies
Components:

- "Create New Distribution" button
- Search and filter (by pharmacy, date, status)
- Distributions Table with columns:
  - Distribution ID
  - Pharmacy Name
  - Medicine Name
  - Quantity
  - Batch Number
  - Distribution Date
  - Status (Pending/In Transit/Delivered/Cancelled)
  - Actions (View Details, Edit, Cancel)
- Distribution Timeline visualization

Page 16: Create Distribution Order Modal

Purpose: Record new medicine distribution
Components:

- Pharmacy dropdown (searchable)
- Medicine dropdown (with current stock display)
- Batch dropdown (filtered by selected medicine)
- Quantity input (with validation against available stock)
- Distribution Date picker
- Expected Delivery Date picker
- Priority dropdown (Normal/Urgent)
- Notes textarea
- Submit button

Page 17: Purchase Order Management (`/admin/purchase-orders`)

Purpose: Manage purchase orders to suppliers
Components:

- "Create Purchase Order" button
- Search and filter options
- Purchase Orders Table with columns:
  - PO Number
  - Supplier Name
  - Order Date
  - Expected Delivery Date
  - Total Amount
  - Status (Draft/Sent/Confirmed/Received/Cancelled)
  - Actions (View, Edit, Mark as Received, Cancel)

Page 18: Create Purchase Order Page

Purpose: Generate new purchase order
Components:

- Supplier dropdown
- Order Date picker
- Expected Delivery Date picker
- Medicine Items Section:
  - Add Medicine button
  - For each item:
    - Medicine dropdown
    - Quantity input
    - Unit Price input
    - Total price (auto-calculated)
    - Remove button
- Total Order Amount (auto-calculated)
- Notes textarea
- Save as Draft button
- Submit to Supplier button

Page 19: Reports & Analytics (`/admin/reports`)

Purpose: Generate comprehensive reports
Components:

- Report Type Selection:
  - Stock Summary Report
  - Expiry Report
  - Distribution Report
  - Supplier Performance Report
  - Financial Report
- Date Range Picker (From-To)
- Filter Options (Medicine, Supplier, Pharmacy)
- "Generate Report" button
- Report Preview Section
- Export Options (PDF, Excel, CSV)
- Scheduled Reports section (set up automatic report generation)

Page 20: AI Demand Forecasting (`/admin/ai-forecast`)

Purpose: View AI-powered predictions
Components:

- Medicine selector
- Time Period selector (Next 7 days, 30 days, 90 days)
- Forecast Chart (showing predicted demand)
- Confidence Level indicator
- Recommended Reorder Quantity
- Historical Data comparison
- "Generate Forecast" button
- Export Forecast button

Page 21: Expiry Alerts Dashboard (`/admin/alerts/expiry`)

Purpose: Monitor medicines nearing expiry
Components:

- Alert Summary Cards:
  - Expired medicines count
  - Expiring in 30 days count
  - Expiring in 90 days count
- Alert Table with columns:
  - Medicine Name
  - Batch Number
  - Expiry Date
  - Days Remaining
  - Quantity
  - Location
  - Urgency Level (Critical/High/Medium)
  - Actions (Mark as Disposed, Transfer to Return)
- Filter by urgency level
- "Send Notification" button

Page 22: Low Stock Alerts Dashboard (`/admin/alerts/low-stock`)

Purpose: Monitor medicines below reorder level
Components:

- Alert Summary
- Low Stock Table with columns:
  - Medicine Name
  - Current Stock
  - Reorder Level
  - Stock Status
  - Last Supplied Date
  - Recommended Action
  - Actions (Create Purchase Order, Adjust Reorder Level)

Page 23: User Management (`/admin/users`)

Purpose: Manage system users and roles
Components:

- "Add New User" button
- Search bar
- Users Table with columns:
  - Full Name
  - Email
  - Phone
  - Role (Admin/Supplier/Pharmacy)
  - Organization

    - Status (Active/Inactive)
    - Last Login
    - Actions (View, Edit, Deactivate, Reset Password)

Page 24: System Settings (`/admin/settings`)

Purpose: Configure system parameters
Components:

- General Settings Section:
    - System Name
    - Contact Email
    - Phone Number
    - Address
- Notification Settings:
    - Email notifications toggle
    - SMS notifications toggle
    - Alert thresholds (Low Stock %, Expiry days warning)
- Backup Settings:
    - Auto-backup toggle
    - Backup frequency dropdown
    - Last backup date display
- Security Settings:
    - Password policy configuration
    - Session timeout
- Save Changes button

Page 25: Activity Logs (`/admin/logs`)

Purpose: Audit trail of system activities
Components:

- Date range filter
- User filter
- Activity type filter (Login, Stock Update, Medicine Added, etc.)
- Logs Table with columns:
    - Timestamp
    - User
    - Activity Type
    - Description
    - IP Address
    - Status
- Export logs button
- Search functionality

1.3 Supplier Dashboard Pages (Role: SUPPLIER)

Page 26: Supplier Dashboard Home (`/supplier/dashboard`)

Purpose: Supplier overview
Components:

- Total Products Supplied card
- Pending Orders card
- Delivered Orders card
- Revenue This Month card
- Recent Purchase Orders list

- Stock Alerts for supplied medicines
- Order Status Chart
- Quick Actions (View Orders, Update Stock)

Page 27: My Products (`/supplier/products`)

Purpose: Manage supplied medicines
Components:

- "Add New Product" button
- Products Table with columns:
  - Medicine Name
  - Category
  - Current Stock
  - Price
  - Last Supply Date
  - Status
  - Actions (Edit, View History)

Page 28: Purchase Orders (`/supplier/orders`)

Purpose: View and manage received orders
Components:

- Filter by status
- Orders Table with columns:
  - PO Number
  - Order Date
  - Medicines Ordered
  - Quantity
  - Total Amount
  - Delivery Date
  - Status
  - Actions (View Details, Confirm, Mark as Shipped)

Page 29: Order Details Page (`/supplier/orders/:id`)

Purpose: Detailed view of single order
Components:

- Order Information (PO Number, Date, Hospital name)
- Ordered Items Table
- Delivery Information
- Status Timeline
- "Confirm Order" button (if pending)
- "Mark as Shipped" button (if confirmed)
- Upload Delivery Receipt
- Notes section

Page 30: Delivery Management (`/supplier/deliveries`)

Purpose: Track deliveries
Components:

- Deliveries Table with columns:
  - Delivery ID
  - Order Number
  - Delivery Date

- Pharmacy Name
- Items Count
- Status
- Tracking Info
- Actions (View, Update Status)

Page 31: Supplier Profile (`/supplier/profile`)

Purpose: Manage supplier account
Components:

- Company Information section (editable)
- Contact Details
- License Information
- Bank Details for payments
- Password Change section
- Supplied Medicines list
- Performance Metrics (On-time delivery rate, Rating)
- Update button

1.4 Pharmacy/Distributor Dashboard Pages (Role: PHARMACY)

Page 32: Pharmacy Dashboard Home (`/pharmacy/dashboard`)

Purpose: Pharmacy overview
Components:

- Current Stock Value card
- Low Stock Items card
- Orders Placed card
- Medicines Near Expiry card
- Quick Stock Search
- Recent Distributions received
- Stock Level Chart
- Expiry Timeline

Page 33: My Inventory (`/pharmacy/inventory`)

Purpose: View pharmacy's medicine stock
Components:

- Search and filter options
- Inventory Table with columns:
  - Medicine Name
  - Category
  - Batch Number
  - Quantity
  - Expiry Date
  - Status
  - Actions (View Details, Request Restock)
- Low Stock Alert section
- Export Inventory button

Page 34: Request Distribution (`/pharmacy/request`)

Purpose: Request medicines from main stock
Components:

- Medicine selector (searchable)
- Current Stock display
- Quantity Requested input
- Priority selector
- Reason/Notes textarea
- Submit Request button
- My Pending Requests section

Page 35: My Orders (`/pharmacy/orders`)

Purpose: Track distribution requests
Components:

- Filter by status
- Orders Table with columns:
  - Order ID
  - Medicines Requested
  - Quantity
  - Request Date
  - Expected Delivery
  - Status (Pending/Approved/In Transit/Delivered/Rejected)
  - Actions (View, Cancel if pending)

Page 36: Received Distributions (`/pharmacy/received`)

Purpose: Record received medicines
Components:

- Distributions Table
- "Confirm Receipt" action
- Quality Check form (condition, quantity verification)
- Report Issues button

Page 37: Expiry Management (`/pharmacy/expiry`)

Purpose: Monitor pharmacy's expiring medicines
Components:

- Expiry Alert Summary
- Medicines Table with expiry dates
- Filter by urgency
- "Report for Return" button
- Disposal Record section

Page 38: Pharmacy Profile (`/pharmacy/profile`)

Purpose: Manage pharmacy account
Components:

- Pharmacy Information (editable)
- License Details
- Contact Information
- Manager Details
- Password Change
- Operating Hours
- Update button

1.5 Common Pages (All Authenticated Users)

Page 39: Notifications Center (`/notifications`)

Purpose: View all system notifications
Components:

- Tabs: All, Unread, Alerts, Orders
- Notifications List with:
  - Timestamp
  - Notification Type icon
  - Message
  - Action button (if applicable)
  - Mark as Read
- Filter options
- Clear All button

Page 40: Help & Documentation (`/help`)

Purpose: User guidance
Components:

- Search Help Topics
- FAQ Section
- Video Tutorials
- User Manual (PDF download)
- Contact Support form
- System Status indicator

Page 41: Profile Settings (`/profile`)

Purpose: User account management
Components:

- Profile Picture upload
- Personal Information (editable)
- Email & Phone (with verification)
- Password Change section
- Notification Preferences
- Language Selection (English/Nepali)
- Theme Selection (Light/Dark)
- Save Changes button

---

2. BACKEND API REQUIREMENTS

2.1 Authentication & Authorization APIs

API 1: User Registration

Endpoint: `POST /api/auth/register`
Request Body:


json

```
{
  "fullName": "string",
  "email": "string",
  "password": "string",
  "phone": "string",
  "role": "ADMIN|SUPPLIER|PHARMACY",
  "organizationName": "string",
  "licenseNumber": "string",
  "address": "string"
}
```

Response:

json

```
{
  "success": true,
  "message": "Registration successful. Please verify your email.",
  "userId": "string"
}
```

Code Generation Prompt:

```
Create a Spring Boot REST endpoint for user registration with the following requirements:
- Endpoint: POST /api/auth/register
- Validate all input fields (email format, password strength min 8 chars)
- Hash password using BCryptPasswordEncoder
- Assign role based on input (ADMIN, SUPPLIER, PHARMACY)
- Save user to MySQL database using JPA
- Send verification email using JavaMailSender
- Return success response with userId
- Handle duplicate email exception
- Include proper error handling and validation annotations
```

API 2: User Login

Endpoint: `POST /api/auth/login`
Request Body:

json

```
{
  "email": "string",
  "password": "string"
}
```

Response:

json

```
{
  "success": true,
  "token": "jwt_token_string",
  "refreshToken": "refresh_token_string",
  "user": {
    "id": "string",
    "fullName": "string",
    "email": "string",
    "role": "string",
    "organizationName": "string"
  },
  "expiresIn": 3600
}
```

Code Generation Prompt:

```
Create a Spring Boot login endpoint with JWT authentication:
- Endpoint: POST /api/auth/login
- Validate email and password against database
- Use Spring Security for authentication
- Generate JWT token with user details and role
- Generate refresh token
- Set token expiration to 1 hour
- Return user object without password
- Include proper error handling for invalid credentials
- Log login attempts for security audit
```

API 3: Refresh Token

Endpoint: `POST /api/auth/refresh`
Request Body:


json

```
{
  "refreshToken": "string"
}
```

Response:


json

```
{
  "token": "new_jwt_token",
  "expiresIn": 3600
}
```

Code Generation Prompt:

```
Create Spring Boot refresh token endpoint:
- Endpoint: POST /api/auth/refresh
- Validate refresh token
- Generate new JWT access token
- Return new token with expiration time
- Handle invalid/expired refresh token errors
```

API 4: Logout

Endpoint: `POST /api/auth/logout`
Headers: `Authorization: Bearer {token}`
Response:


json

```
{
  "success": true,
  "message": "Logged out successfully"
}
```

API 5: Forgot Password

Endpoint: `POST /api/auth/forgot-password`
Request Body:


json

```
{
  "email": "string"
}
```

Response:


json

```
{
  "success": true,
  "message": "Password reset link sent to email"
}
```

API 6: Reset Password

Endpoint: `POST /api/auth/reset-password`
Request Body:


json

```
{
  "token": "reset_token",
  "newPassword": "string"
}
```

2.2 Medicine Management APIs

API 7: Get All Medicines (Paginated)

Endpoint: `GET /api/medicines?page=0&size=10&search=&category=&sort=name`
Headers: `Authorization: Bearer {token}`
Response:

json


```
{
  "content": [
    {
      "id": "string",
      "name": "string",
      "genericName": "string",
      "category": "string",
      "manufacturer": "string",
      "currentStock": 0,
      "unitPrice": 0.00,
      "reorderLevel": 0,
      "status": "AVAILABLE|LOW_STOCK|OUT_OF_STOCK",
      "imageUrl": "string",
      "createdAt": "datetime",
      "updatedAt": "datetime"
    }
  ],
  "totalPages": 5,
  "totalElements": 50,
  "currentPage": 0
}
```


Code Generation Prompt:

Create Spring Boot GET endpoint for fetching medicines with pagination:
- Endpoint: GET /api/medicines
- Support query parameters: page, size, search, category, sort
- Use Spring Data JPA Pageable
- Implement search functionality (name, genericName)
- Filter by category if provided
- Sort by specified field (default: name)
- Calculate stock status dynamically (compare with reorderLevel)
- Return paginated response
- Secure with JWT authentication using @PreAuthorize
- Only accessible by ADMIN and PHARMACY roles

API 8: Get Medicine by ID

Endpoint: `GET /api/medicines/{id}`
Response:

json


```
{
  "id": "string",
  "name": "string",
  "genericName": "string",
  "category": "string",
  "manufacturer": "string",
  "description": "string",
  "currentStock": 0,
  "unitPrice": 0.00,
  "reorderLevel": 0,
  "suppliers": ["supplier1", "supplier2"],
  "batches": [
    {
      "batchNumber": "string",
      "quantity": 0,
      "expiryDate": "date"
    }
  ],
  "status": "string",
  "imageUrl": "string"
}
```


API 9: Create Medicine

Endpoint: `POST /api/medicines`
Request Body:


json

```
{
  "name": "string",
  "genericName": "string",
  "category": "string",
  "manufacturer": "string",
  "description": "string",
  "unitPrice": 0.00,
  "reorderLevel": 0,
  "supplierIds": ["id1", "id2"],
  "imageUrl": "string"
}
```

Response:


json

```
{
  "success": true,
  "message": "Medicine created successfully",
  "medicineId": "string"
}
```

Code Generation Prompt:

```
Create Spring Boot POST endpoint to add new medicine:
- Endpoint: POST /api/medicines
- Validate all required fields using @Valid annotation
- Create Medicine entity and save to database
- Link suppliers using Many-to-Many relationship
- Initialize currentStock as 0
- Calculate initial status based on stock
- Upload image to cloud storage if provided
- Return success response with generated medicineId
- Only ADMIN can access (use @PreAuthorize("hasRole('ADMIN')"))
- Handle duplicate medicine name exception
- Log medicine creation activity
```

API 10: Update Medicine

Endpoint: `PUT /api/medicines/{id}`
Request Body: (same as create)
Response:


json

```
{
  "success": true,
  "message": "Medicine updated successfully"
}
```

API 11: Delete Medicine

Endpoint: `DELETE /api/medicines/{id}`
Response:


json

```
{
  "success": true,
  "message": "Medicine deleted successfully"
}
```

Code Generation Prompt:

```
Create Spring Boot DELETE endpoint for medicine:
- Endpoint: DELETE /api/medicines/{id}
- Check if medicine exists
- Check if medicine has active stock (prevent deletion if stock > 0)
- Check if medicine is in any pending orders
- Perform soft delete (set active=false) instead of hard delete
- Only ADMIN role can access
- Return appropriate error if medicine cannot be deleted
- Log deletion activity
```

API 12: Bulk Upload Medicines

Endpoint: `POST /api/medicines/bulk-upload`
Request: Multipart file (CSV/Excel)
Response:

json

```
{
  "success": true,
  "message": "Processed 50 records",
  "successCount": 45,
  "failureCount": 5,
  "errors": [
    {
      "row": 3,
      "error": "Invalid category"
    }
  ]
}
```

2.3 Batch Management APIs

API 13: Get All Batches

Endpoint: `GET /api/batches?page=0&size=10&medicineId=&supplierId=&expiryStatus=`
Response:

json

```
{
 "content": [
  {
   "id": "string",
   "batchNumber": "string",
   "medicine": {
    "id": "string",
    "name": "string"
   },
   "quantity": 0,
   "manufacturingDate": "date",
   "expiryDate": "date",
   "daysToExpiry": 45,
   "supplier": {
    "id": "string",
    "name": "string"
   },
   "purchasePrice": 0.00,
   "storageLocation": "string",
   "status": "ACTIVE|NEAR_EXPIRY|EXPIRED",
   "createdAt": "datetime"
  }
 ],
 "totalElements": 100
}
```


Code Generation Prompt:

```
Create Spring Boot GET endpoint for batches with filtering:
- Endpoint: GET /api/batches
- Support pagination and filters (medicineId, supplierId, expiryStatus)
- Calculate daysToExpiry dynamically (expiryDate - current date)
- Determine status based on daysToExpiry:
  * EXPIRED if daysToExpiry < 0
  * NEAR_EXPIRY if daysToExpiry < 90
  * ACTIVE otherwise
- Join with Medicine and Supplier tables
- Sort by expiryDate ascending by default
- Accessible by ADMIN and PHARMACY roles
```

API 14: Create Batch

Endpoint: `POST /api/batches`
Request Body:


json

```
{
  "batchNumber": "string",
  "medicineId": "string",
  "quantity": 0,
  "manufacturingDate": "date",
  "expiryDate": "date",
  "supplierId": "string",
  "purchasePrice": 0.00,
  "storageLocation": "string",
  "notes": "string"
}
```

Code Generation Prompt:

```
Create Spring Boot POST endpoint to add new batch:
- Endpoint: POST /api/batches
- Validate batch number is unique
- Validate expiry date is after manufacturing date
- Fetch Medicine entity and add quantity to currentStock
- Create Batch entity with relationships
- Update medicine stock level
- Trigger stock level recalculation
- Only ADMIN and SUPPLIER can access
- Log batch creation with audit trail
- Send notification if medicine was previously out of stock
```

API 15: Update Batch

Endpoint: `PUT /api/batches/{id}`

API 16: Delete Batch

Endpoint: `DELETE /api/batches/{id}`

API 17: Get Expiring Batches

Endpoint: `GET /api/batches/expiring?days=30`
Response: List of batches expiring within specified days

2.4 Stock Management APIs

API 18: Get Stock Summary

Endpoint: `GET /api/stock/summary`
Response:


json


```
{
  "totalMedicines": 150,
  "totalStockValue": 500000.00,
  "lowStockCount": 12,
  "outOfStockCount": 5,
  "nearExpiryCount": 8,
  "expiredCount": 3
}
```


API 19: Get Stock by Medicine

Endpoint: `GET /api/stock/medicine/{medicineId}`
Response:


json

```
{
  "medicineId": "string",
  "medicineName": "string",
  "currentStock": 500,
  "reorderLevel": 100,
  "stockStatus": "ADEQUATE|LOW|OUT",
  "batches": [
    {
      "batchNumber": "string",
      "quantity": 200,
      "expiryDate": "date"
    }
  ],
  "locations": [
    {
      "location": "Main Warehouse",
      "quantity": 300
    },
    {
      "location": "Pharmacy Branch 1",
      "quantity": 200
    }
  ]
}
```


API 20: Adjust Stock

Endpoint: `POST /api/stock/adjust`
Request Body:

json


```
{
  "medicineId": "string",
  "batchNumber": "string",
  "adjustmentType": "ADD|REMOVE",
  "quantity": 0,
  "reason": "RECEIVED|DAMAGED|EXPIRED|SOLD|RETURNED",
  "notes": "string",
  "date": "date"
}
```

Code Generation Prompt:

```
Create Spring Boot POST endpoint for stock adjustment:
- Endpoint: POST /api/stock/adjust
- Validate medicine and batch exist
- If ADD: increase batch quantity and medicine stock
- If REMOVE: decrease batch quantity, prevent negative stock
- Create StockTransaction record for audit trail
- Update medicine's lastUpdated timestamp
- Recalculate stock status
- Trigger low stock alert if necessary
- Only ADMIN can access
- Return updated stock details
```

API 21: Transfer Stock

Endpoint: `POST /api/stock/transfer`
Request Body:

json

```
{
  "medicineId": "string",
  "batchNumber": "string",
  "fromLocation": "string",
  "toLocation": "string",
  "quantity": 0,
  "notes": "string"
}
```

API 22: Get Low Stock Medicines

Endpoint: `GET /api/stock/low-stock`
Response: List of medicines below reorder level

API 23: Get Stock History

Endpoint: `GET /api/stock/history/{medicineId}?startDate=&endDate=`
Response: Stock transaction history for a medicine

2.5 Supplier Management APIs

API 24: Get All Suppliers

Endpoint: `GET /api/suppliers?page=0&size=10&search=&status=`
Response:


json


```
{
  "content": [
    {
      "id": "string",
      "companyName": "string",
      "contactPerson": "string",
      "email": "string",
      "phone": "string",
      "address": "string",
      "licenseNumber": "string",
      "status": "ACTIVE|INACTIVE",
      "rating": 4.5,
      "totalSupplies": 50,
      "medicinesSupplied": ["Medicine1", "Medicine2"],
      "createdAt": "datetime"
    }
  ],
  "totalElements": 25
}
```


Code Generation Prompt:

```
Create Spring Boot GET endpoint for suppliers with pagination:
- Endpoint: GET /api/suppliers
- Support search by company name, contact person
- Filter by status (ACTIVE/INACTIVE)
- Include count of total supplies delivered
- Calculate average rating from supplier_ratings table
- Use pagination with Spring Data JPA
- Accessible by ADMIN and PHARMACY roles
- Include list of medicines supplied (Many-to-Many relationship)
```

API 25: Get Supplier by ID

Endpoint: `GET /api/suppliers/{id}`
Response:

json


```
{
 "id": "string",
 "companyName": "string",
 "contactPerson": "string",
 "email": "string",
 "phone": "string",
 "address": "string",
 "licenseNumber": "string",
 "medicinesSupplied": [
  {
   "id": "string",
   "name": "string",
   "price": 0.00
  }
 ],
 "paymentTerms": "string",
 "status": "ACTIVE|INACTIVE",
 "notes": "string",
 "bankDetails": {
  "accountName": "string",
  "accountNumber": "string",
  "bankName": "string"
 },
 "performance": {
  "totalOrders": 100,
  "completedOrders": 95,
  "onTimeDeliveryRate": 92.5,
  "averageRating": 4.5
 }
}
```


API 26: Create Supplier

Endpoint: `POST /api/suppliers`
Request Body:

json

```
{
  "companyName": "string",
  "contactPerson": "string",
  "email": "string",
  "phone": "string",
  "address": "string",
  "licenseNumber": "string",
  "medicineIds": ["id1", "id2"],
  "paymentTerms": "string",
  "bankDetails": {
    "accountName": "string",
    "accountNumber": "string",
    "bankName": "string"
  },
  "notes": "string"
}
```

Response:

json

```
{
  "success": true,
  "message": "Supplier created successfully",
  "supplierId": "string"
}
```

Code Generation Prompt:

Create Spring Boot POST endpoint to add new supplier:
- Endpoint: POST /api/suppliers
- Validate email format and phone number
- Check for duplicate license number
- Create Supplier entity with status=ACTIVE by default
- Link medicines using Many-to-Many relationship
- Hash and store bank details securely
- Only ADMIN can access
- Send welcome email to supplier
- Create default user account for supplier portal access
- Log supplier creation
- Return success with generated supplierId

API 27: Update Supplier

Endpoint: `PUT /api/suppliers/{id}`
Request Body: (same structure as create)

API 28: Delete Supplier

Endpoint: `DELETE /api/suppliers/{id}`

API 29: Get Supplier Performance

Endpoint: `GET /api/suppliers/{id}/performance`
Response:


json

```
{
  "supplierId": "string",
  "totalOrders": 100,
  "completedOrders": 95,
  "pendingOrders": 3,
  "cancelledOrders": 2,
  "onTimeDeliveryRate": 92.5,
  "averageDeliveryTime": 3.5,
  "averageRating": 4.5,
  "recentOrders": [
    {
      "orderId": "string",
      "orderDate": "date",
      "deliveryDate": "date",
      "status": "string",
      "amount": 0.00
    }
  ]
}
```


API 30: Rate Supplier

Endpoint: `POST /api/suppliers/{id}/rate`
Request Body:


json


```
{
  "rating": 4.5,
  "comment": "string",
  "orderId": "string"
}
```

2.6 Pharmacy/Distributor Management APIs

API 31: Get All Pharmacies

Endpoint: `GET /api/pharmacies?page=0&size=10&search=&status=`
Response:


json

```
{
 "content": [
  {
   "id": "string",
   "pharmacyName": "string",
   "ownerName": "string",
   "email": "string",
   "phone": "string",
   "address": "string",
   "licenseNumber": "string",
   "status": "ACTIVE|INACTIVE",
   "registrationDate": "date",
   "totalOrders": 50,
   "currentStockValue": 100000.00
  }
 ],
 "totalElements": 30
}
```


API 32: Get Pharmacy by ID

Endpoint: `GET /api/pharmacies/{id}`

API 33: Create Pharmacy

Endpoint: `POST /api/pharmacies`
Request Body:


json


```
{
 "pharmacyName": "string",
 "ownerName": "string",
 "managerName": "string",
 "email": "string",
 "phone": "string",
 "address": "string",
 "licenseNumber": "string",
 "operatingHours": "string"
}
```


Code Generation Prompt:

Create Spring Boot POST endpoint to register new pharmacy:
- Endpoint: POST /api/pharmacies
- Validate license number uniqueness
- Create Pharmacy entity with status=ACTIVE
- Create user account for pharmacy manager
- Send credentials via email
- Only ADMIN can access
- Log pharmacy registration
- Return success with pharmacyId

API 34: Update Pharmacy

Endpoint: `PUT /api/pharmacies/{id}`

API 35: Deactivate Pharmacy

Endpoint: `PUT /api/pharmacies/{id}/deactivate`

API 36: Get Pharmacy Inventory

Endpoint: `GET /api/pharmacies/{id}/inventory`
Response:

json

```
{
  "pharmacyId": "string",
  "pharmacyName": "string",
  "inventory": [
    {
      "medicineId": "string",
      "medicineName": "string",
      "batchNumber": "string",
      "quantity": 0,
      "expiryDate": "date",
      "status": "string"
    }
  ],
  "totalValue": 100000.00,
  "lowStockCount": 5
}
```

2.7 Distribution Management APIs

API 37: Get All Distributions

Endpoint: `GET /api/distributions?page=0&size=10&pharmacyId=&status=&startDate=&endDate=`
Response:

json

```
{
 "content": [
  {
   "id": "string",
   "distributionNumber": "string",
   "pharmacy": {
    "id": "string",
    "name": "string"
   },
   "medicines": [
    {
     "medicineId": "string",
     "medicineName": "string",
     "batchNumber": "string",
     "quantity": 0
    }
   ],
   "distributionDate": "date",
   "expectedDeliveryDate": "date",
   "actualDeliveryDate": "date",
   "status": "PENDING|IN_TRANSIT|DELIVERED|CANCELLED",
   "priority": "NORMAL|URGENT",
   "totalValue": 0.00,
   "createdBy": "string",
   "notes": "string"
  }
 ],
 "totalElements": 75
}
```


Code Generation Prompt:

Create Spring Boot GET endpoint for distributions:
- Endpoint: GET /api/distributions
- Support multiple filters: pharmacyId, status, date range
- Join with Pharmacy and Medicine tables
- Calculate totalValue from medicine quantities and prices
- Sort by distributionDate descending
- ADMIN sees all distributions
- PHARMACY sees only their distributions
- Use role-based filtering with Spring Security
- Return paginated results

API 38: Get Distribution by ID

Endpoint: `GET /api/distributions/{id}`
Response:

json

```
{
 "id": "string",
 "distributionNumber": "string",
 "pharmacy": {
  "id": "string",
  "name": "string",
  "address": "string",
  "contactPerson": "string",
  "phone": "string"
 },
 "items": [
  {
   "medicineId": "string",
   "medicineName": "string",
   "batchNumber": "string",
   "quantity": 0,
   "unitPrice": 0.00,
   "totalPrice": 0.00,
   "expiryDate": "date"
  }
 ],
 "distributionDate": "date",
 "expectedDeliveryDate": "date",
 "actualDeliveryDate": "date",
 "status": "string",
 "priority": "string",
 "totalValue": 0.00,
 "shippingAddress": "string",
 "trackingNumber": "string",
 "notes": "string",
 "timeline": [
  {
   "status": "CREATED",
   "timestamp": "datetime",
   "note": "Distribution order created"
  },
  {
   "status": "IN_TRANSIT",
   "timestamp": "datetime",
   "note": "Order dispatched"
  }
 ],
 "createdBy": "string",
```

```
  "createdAt": "datetime"
}
```

API 39: Create Distribution

Endpoint: `POST /api/distributions`
Request Body:


json

```
{
  "pharmacyId": "string",
  "items": [
    {
      "medicineId": "string",
      "batchNumber": "string",
      "quantity": 0
    }
  ],
  "distributionDate": "date",
  "expectedDeliveryDate": "date",
  "priority": "NORMAL|URGENT",
  "notes": "string"
}
```

Response:


json

```
{
  "success": true,
  "message": "Distribution created successfully",
  "distributionId": "string",
  "distributionNumber": "string"
}
```

Code Generation Prompt:

Create Spring Boot POST endpoint for creating distribution:
- Endpoint: POST /api/distributions
- Validate pharmacy exists and is active
- For each item, validate:
  * Medicine exists
  * Batch exists and has sufficient quantity
  * Stock is available
- Generate unique distribution number (format: DIST-YYYYMMDD-XXXX)
- Create Distribution entity with status=PENDING
- Create DistributionItem entities for each medicine
- Reduce stock quantities from batches
- Update medicine current stock
- Create stock transaction records
- Calculate total value
- Send notification to pharmacy
- Only ADMIN can create distributions
- Return success with distribution details

API 40: Update Distribution Status

Endpoint: `PUT /api/distributions/{id}/status`
Request Body:


json

```
{
  "status": "IN_TRANSIT|DELIVERED|CANCELLED",
  "note": "string",
  "actualDeliveryDate": "date"
}
```

Code Generation Prompt:

Create Spring Boot PUT endpoint to update distribution status:
- Endpoint: PUT /api/distributions/{id}/status
- Validate status transition is allowed:
  * PENDING -> IN_TRANSIT, CANCELLED
  * IN_TRANSIT -> DELIVERED, CANCELLED
  * DELIVERED -> (no further changes)
- Update distribution status and actualDeliveryDate if DELIVERED
- Add entry to timeline/history
- If CANCELLED, restore stock quantities to batches
- Send status update notification to pharmacy
- ADMIN can update any status
- PHARMACY can only mark as DELIVERED (confirming receipt)
- Log status change activity

API 41: Cancel Distribution

Endpoint: `DELETE /api/distributions/{id}`

API 42: Get Distribution Statistics

Endpoint: `GET /api/distributions/statistics?startDate=&endDate=`
Response:

json

```
{
  "totalDistributions": 100,
  "pendingDistributions": 10,
  "inTransitDistributions": 5,
  "deliveredDistributions": 80,
  "cancelledDistributions": 5,
  "totalValue": 500000.00,
  "averageDeliveryTime": 2.5,
  "onTimeDeliveryRate": 95.0,
  "topPharmacies": [
    {
      "pharmacyName": "string",
      "orderCount": 20
    }
  ],
  "topMedicines": [
    {
      "medicineName": "string",
      "distributedQuantity": 1000
    }
  ]
}
```

2.8 Purchase Order Management APIs

API 43: Get All Purchase Orders

Endpoint: `GET /api/purchase-orders?page=0&size=10&supplierId=&status=&startDate=&endDate=`
Response:

json

```
{
  "content": [
    {
      "id": "string",
      "poNumber": "string",
      "supplier": {
        "id": "string",
        "name": "string"
      },
      "orderDate": "date",
      "expectedDeliveryDate": "date",
      "actualDeliveryDate": "date",
      "totalAmount": 0.00,
      "status": "DRAFT|SENT|CONFIRMED|RECEIVED|CANCELLED",
      "itemCount": 5,
      "createdBy": "string"
    }
  ],
  "totalElements": 50
}
```


API 44: Get Purchase Order by ID

Endpoint: `GET /api/purchase-orders/{id}`
Response:

json

```
{
 "id": "string",
 "poNumber": "string",
 "supplier": {
  "id": "string",
  "name": "string",
  "contactPerson": "string",
  "email": "string",
  "phone": "string"
 },
 "orderDate": "date",
 "expectedDeliveryDate": "date",
 "actualDeliveryDate": "date",
 "items": [
  {
   "medicineId": "string",
   "medicineName": "string",
   "quantity": 0,
   "unitPrice": 0.00,
   "totalPrice": 0.00
  }
 ],
 "subtotal": 0.00,
 "tax": 0.00,
 "totalAmount": 0.00,
 "status": "string",
 "paymentTerms": "string",
 "notes": "string",
 "timeline": [
  {
   "status": "DRAFT",
   "timestamp": "datetime",
   "updatedBy": "string"
  }
 ]
}
```


API 45: Create Purchase Order

Endpoint: `POST /api/purchase-orders`
Request Body:

json

```
{
  "supplierId": "string",
  "orderDate": "date",
  "expectedDeliveryDate": "date",
  "items": [
    {
      "medicineId": "string",
      "quantity": 0,
      "unitPrice": 0.00
    }
  ],
  "paymentTerms": "string",
  "notes": "string",
  "status": "DRAFT|SENT"
}
```

Response:

json

```
{
  "success": true,
  "message": "Purchase order created successfully",
  "poId": "string",
  "poNumber": "string"
}
```

Code Generation Prompt:

```
Create Spring Boot POST endpoint for purchase order creation:
- Endpoint: POST /api/purchase-orders
- Validate supplier exists and is active
- Validate all medicines exist
- Generate unique PO number (format: PO-YYYYMMDD-XXXX)
- Create PurchaseOrder entity
- Create PurchaseOrderItem entities for each medicine
- Calculate subtotal, tax (if applicable), and total
- If status is SENT, send email to supplier with PO details
- Only ADMIN can create purchase orders
- Log PO creation
- Add initial entry to timeline
- Return success with PO details
```

API 46: Update Purchase Order

Endpoint: `PUT /api/purchase-orders/{id}`
Request Body: (same structure as create)
Note: Can only update if status is DRAFT

API 47: Send Purchase Order to Supplier

Endpoint: `POST /api/purchase-orders/{id}/send`
Response:


json

```
{
  "success": true,
  "message": "Purchase order sent to supplier"
}
```

API 48: Confirm Purchase Order (Supplier)

Endpoint: `POST /api/purchase-orders/{id}/confirm`
Request Body:


json

```
{
  "estimatedDeliveryDate": "date",
  "notes": "string"
}
```

Code Generation Prompt:

```
Create Spring Boot POST endpoint for supplier to confirm PO:
- Endpoint: POST /api/purchase-orders/{id}/confirm
- Validate PO status is SENT
- Update status to CONFIRMED
- Update estimatedDeliveryDate if provided
- Add timeline entry
- Send confirmation notification to admin
- Only SUPPLIER role can confirm their own POs
- Use @PreAuthorize with SpEL to check supplier ownership
```

API 49: Mark Purchase Order as Received

Endpoint: `POST /api/purchase-orders/{id}/receive`
Request Body:

json

```
{
  "actualDeliveryDate": "date",
  "receivedItems": [
    {
      "medicineId": "string",
      "batchNumber": "string",
      "receivedQuantity": 0,
      "manufacturingDate": "date",
      "expiryDate": "date",
      "damageQuantity": 0,
      "notes": "string"
    }
  ],
  "invoiceNumber": "string",
  "invoiceAmount": 0.00
}
```

Code Generation Prompt:

```
Create Spring Boot POST endpoint to receive purchase order:
- Endpoint: POST /api/purchase-orders/{id}/receive
- Validate PO status is CONFIRMED
- For each received item:
  * Create new Batch entity
  * Add receivedQuantity to medicine stock
  * Create stock transaction record
  * If damageQuantity > 0, record in separate damage report
- Update PO status to RECEIVED
- Set actualDeliveryDate
- Calculate delivery time and update supplier performance metrics
- Add timeline entry
- Send receipt confirmation to supplier
- Only ADMIN can mark as received
- Trigger low stock alert resolution if applicable
```

API 50: Cancel Purchase Order

Endpoint: `DELETE /api/purchase-orders/{id}`
Request Body:

json

```
{
 "reason": "string"
}
```

2.9 Alert & Notification APIs

API 51: Get All Notifications

Endpoint: `GET /api/notifications?page=0&size=20&type=&read=`
Response:

json

```
{
 "content": [
  {
   "id": "string",
   "type": "LOW_STOCK|EXPIRY|ORDER|SYSTEM",
   "title": "string",
   "message": "string",
   "relatedEntityType": "MEDICINE|BATCH|DISTRIBUTION|PO",
   "relatedEntityId": "string",
   "priority": "LOW|MEDIUM|HIGH|CRITICAL",
   "isRead": false,
   "createdAt": "datetime",
   "actionUrl": "string"
  }
 ],
 "unreadCount": 15,
 "totalElements": 50
}
```

Code Generation Prompt:

Create Spring Boot GET endpoint for user notifications:
- Endpoint: GET /api/notifications
- Filter by type (LOW_STOCK, EXPIRY, ORDER, SYSTEM)
- Filter by read status
- Return notifications for authenticated user only
- Sort by createdAt descending (newest first)
- Include unreadCount in response
- Support pagination
- Include actionUrl for quick navigation

API 52: Mark Notification as Read

Endpoint: `PUT /api/notifications/{id}/read`

API 53: Mark All Notifications as Read

Endpoint: `PUT /api/notifications/read-all`

API 54: Get Expiry Alerts

Endpoint: `GET /api/alerts/expiry?days=30`
Response:

json


```
{
  "criticalCount": 5,
  "warningCount": 12,
  "alerts": [
    {
      "batchId": "string",
      "batchNumber": "string",
      "medicineId": "string",
      "medicineName": "string",
      "quantity": 0,
      "expiryDate": "date",
      "daysToExpiry": 15,
      "urgencyLevel": "CRITICAL|HIGH|MEDIUM",
      "location": "string",
      "recommendedAction": "RETURN|DISPOSE|CLEARANCE_SALE"
    }
  ]
}
```

Code Generation Prompt:

Create Spring Boot GET endpoint for expiry alerts:
- Endpoint: GET /api/alerts/expiry
- Accept 'days' parameter (default: 90)
- Query batches where expiryDate <= (currentDate + days)
- Calculate daysToExpiry
- Determine urgencyLevel:
  * CRITICAL: <= 30 days
  * HIGH: 31-60 days
  * MEDIUM: 61-90 days
- Suggest recommendedAction based on urgency and quantity
- Sort by daysToExpiry ascending
- Include medicine details with JOIN
- Accessible by ADMIN and PHARMACY roles

API 55: Get Low Stock Alerts

Endpoint: `GET /api/alerts/low-stock`
Response:

json

```
{
 "totalAlerts": 10,
 "criticalCount": 3,
 "alerts": [
  {
   "medicineId": "string",
   "medicineName": "string",
   "currentStock": 50,
   "reorderLevel": 100,
   "shortfallQuantity": 50,
   "urgencyLevel": "CRITICAL|HIGH",
   "lastSupplyDate": "date",
   "suggestedReorderQuantity": 200,
   "estimatedStockoutDate": "date"
  }
 ]
}
```


Code Generation Prompt:

```
Create Spring Boot GET endpoint for low stock alerts:
- Endpoint: GET /api/alerts/low-stock
- Query medicines where currentStock <= reorderLevel
- Calculate shortfallQuantity (reorderLevel - currentStock)
- Determine urgencyLevel:
  * CRITICAL: currentStock <= (reorderLevel * 0.5)
  * HIGH: currentStock <= reorderLevel
- Calculate suggestedReorderQuantity (consider historical usage)
- Estimate stockoutDate based on average daily usage
- Sort by urgencyLevel, then shortfallQuantity
- Include last purchase order date
- Only ADMIN can access
```

API 56: Create Manual Alert

Endpoint: `POST /api/alerts`
Request Body:

json

```
{
  "type": "LOW_STOCK|EXPIRY|CUSTOM",
  "title": "string",
  "message": "string",
  "relatedEntityType": "MEDICINE|BATCH",
  "relatedEntityId": "string",
  "priority": "LOW|MEDIUM|HIGH|CRITICAL",
  "recipientRoles": ["ADMIN", "PHARMACY"]
}
```

API 57: Dismiss Alert

Endpoint: `DELETE /api/alerts/{id}`

2.10 Report & Analytics APIs

API 58: Get Dashboard Statistics

Endpoint: `GET /api/reports/dashboard`
Response:

json

```
{
 "overview": {
  "totalMedicines": 250,
  "totalSuppliers": 30,
  "totalPharmacies": 15,
  "totalStockValue": 1500000.00
 },
 "stock": {
  "adequateStock": 200,
  "lowStock": 35,
  "outOfStock": 15,
  "nearExpiry": 20,
  "expired": 5
 },
 "distributions": {
  "today": 5,
  "thisWeek": 25,
  "thisMonth": 100,
  "pending": 10,
  "inTransit": 8
 },
 "purchaseOrders": {
  "draft": 3,
  "sent": 5,
  "confirmed": 7,
  "received": 50
 },
 "recentActivity": [
  {
   "type": "DISTRIBUTION_CREATED",
   "description": "string",
   "timestamp": "datetime",
   "user": "string"
  }
 ]
}
```


Code Generation Prompt:

Create Spring Boot GET endpoint for dashboard statistics:
- Endpoint: GET /api/reports/dashboard
- Aggregate data from multiple tables:
  * Count medicines by status
  * Calculate total stock value (sum of quantity * price)
  * Count distributions by date ranges and status
  * Count purchase orders by status
  * Fetch recent 10 activities from audit log
- Use native queries or JPQL for complex aggregations
- Cache results for 5 minutes to improve performance
- Role-based data filtering:
  * ADMIN sees all data
  * PHARMACY sees only their distributions
  * SUPPLIER sees only their orders

API 59: Get Stock Report

Endpoint: `GET /api/reports/stock?startDate=&endDate=&category=&format=json`
Response:


json

```
{
  "reportDate": "datetime",
  "dateRange": {
    "from": "date",
    "to": "date"
  },
  "summary": {
    "totalMedicines": 250,
    "totalStockValue": 1500000.00,
    "categoryBreakdown": [
      {
        "category": "Antibiotics",
        "count": 50,
        "value": 300000.00
      }
    ]
  },
  "medicines": [
    {
      "name": "string",
      "category": "string",
      "currentStock": 0,
      "stockValue": 0.00,
      "status": "string",
      "batches": 3,
      "nearestExpiry": "date"
    }
  ],
  "lowStockItems": [],
  "expiringItems": []
}
```


API 60: Get Distribution Report

Endpoint: `GET /api/reports/distribution?startDate=&endDate=&pharmacyId=&format=json`
Response:

json

```
{
 "reportDate": "datetime",
 "dateRange": {
  "from": "date",
  "to": "date"
 },
 "summary": {
  "totalDistributions": 100,
  "totalValue": 500000.00,
  "deliveredCount": 85,
  "pendingCount": 10,
  "cancelledCount": 5,
  "averageDeliveryTime": 2.5
 },
 "pharmacyBreakdown": [
  {
   "pharmacyName": "string",
   "distributionCount": 20,
   "totalValue": 100000.00
  }
 ],
 "medicineBreakdown": [
  {
   "medicineName": "string",
   "totalQuantity": 1000,
   "distributionCount": 15
  }
 ],
 "distributions": []
}
```


API 61: Get Supplier Performance Report

Endpoint: `GET /api/reports/supplier-performance?startDate=&endDate=&supplierId=`
Response:

json

```
{
  "reportDate": "datetime",
  "dateRange": {
    "from": "date",
    "to": "date"
  },
  "suppliers": [
    {
      "supplierId": "string",
      "supplierName": "string",
      "totalOrders": 50,
      "completedOrders": 48,
      "cancelledOrders": 2,
      "totalValue": 500000.00,
      "onTimeDeliveryRate": 95.0,
      "averageDeliveryTime": 3.2,
      "averageRating": 4.5,
      "qualityIssues": 1
    }
  ]
}
```


API 62: Get Financial Report

Endpoint: `GET /api/reports/financial?startDate=&endDate=&type=PURCHASE|DISTRIBUTION`
Response:

json

```
{
 "reportDate": "datetime",
 "dateRange": {
  "from": "date",
  "to": "date"
 },
 "type": "PURCHASE",
 "summary": {
  "totalPurchases": 50,
  "totalAmount": 500000.00,
  "averageOrderValue": 10000.00,
  "taxAmount": 65000.00
 },
 "monthlyBreakdown": [
  {
   "month": "2025-01",
   "orderCount": 15,
   "amount": 150000.00
  }
 ],
 "categoryBreakdown": [
  {
   "category": "Antibiotics",
   "amount": 200000.00,
   "percentage": 40.0
  }
 ],
 "topSuppliers": [
  {
   "supplierName": "string",
   "amount": 100000.00
  }
 ]
}
```


API 63: Get Expiry Report

Endpoint: `GET /api/reports/expiry?days=90&includeExpired=true`
Response:

json

```
{
 "reportDate": "datetime",
 "summary": {
  "expiredCount": 5,
  "expiringCount": 20,
  "totalWastageValue": 50000.00
 },
 "expired": [
  {
   "medicineName": "string",
   "batchNumber": "string",
   "quantity": 0,
   "expiryDate": "date",
   "wastageValue": 0.00,
   "daysExpired": 10
  }
 ],
 "expiring": [
  {
   "medicineName": "string",
   "batchNumber": "string",
   "quantity": 0,
   "expiryDate": "date",
   "daysToExpiry": 30,
   "value": 0.00,
   "urgencyLevel": "string"
  }
 ]
}
```


API 64: Export Report

Endpoint: `GET /api/reports/export?type=STOCK|DISTRIBUTION|EXPIRY&format=PDF|EXCEL|CSV&startDate=&endDate=`
Response: File download (application/pdf, application/vnd.ms-excel, text/csv)

Code Generation Prompt:

Create Spring Boot GET endpoint for report export:
- Endpoint: GET /api/reports/export
- Support multiple report types and formats
- For PDF: use iText or Apache PDFBox library
- For Excel: use Apache POI library
- For CSV: use OpenCSV library
- Generate report data based on type and date range
- Format data according to requested format
- Set appropriate response headers (Content-Type, Content-Disposition)
- Return file as downloadable attachment
- Only ADMIN can export reports
- Log export activity with user and timestamp

2.11 AI/ML Prediction APIs (Flask Microservice)

API 65: Get Demand Forecast

Endpoint: `POST /api/ai/forecast/demand`
Request Body:


json

```
{
  "medicineId": "string",
  "forecastPeriod": 30,
  "historicalDataPoints": 90
}
```

Response:


json

```
{
  "medicineId": "string",
  "medicineName": "string",
  "forecastPeriod": 30,
  "predictions": [
    {
      "date": "2025-11-09",
      "predictedDemand": 150,
      "confidenceInterval": {
        "lower": 120,
        "upper": 180
      },
      "confidence": 0.85
    }
  ],
  "summary": {
    "totalPredictedDemand": 4500,
    "averageDailyDemand": 150,
    "trend": "INCREASING|STABLE|DECREASING",
    "seasonalPattern": true
  },
  "recommendations": {
    "suggestedReorderQuantity": 5000,
    "reorderDate": "2025-11-15",
    "reasoning": "Based on predicted demand trend"
  }
}
```


Code Generation Prompt (Python Flask):

Create Flask REST API endpoint for demand forecasting:
- Endpoint: POST /api/ai/forecast/demand
- Accept medicineId, forecastPeriod, historicalDataPoints
- Fetch historical distribution/sales data from database
- Preprocess data: handle missing values, normalize
- Use time series forecasting (ARIMA, Prophet, or LSTM):
  * Train model on historical data
  * Generate predictions for forecast period
  * Calculate confidence intervals
- Detect trend (increasing/stable/decreasing) using linear regression
- Identify seasonal patterns using autocorrelation
- Calculate suggested reorder quantity based on:
  * Predicted demand
  * Safety stock (2 weeks worth)
  * Current stock level
- Return predictions with confidence scores
- Handle errors gracefully (insufficient data, model failures)
- Cache model for 24 hours to improve performance
- Libraries: Flask, pandas, numpy, scikit-learn, statsmodels, tensorflow

API 66: Get Expiry Risk Prediction

Endpoint: `POST /api/ai/predict/expiry-risk`
Request Body:

json

```
{
  "batchId": "string",
  "currentStock": 0,
  "expiryDate": "date"
}
```

Response:

json

```
{
  "batchId": "string",
  "batchNumber": "string",
  "medicineName": "string",
  "expiryRisk": "LOW|MEDIUM|HIGH|CRITICAL",
  "riskScore": 0.75,
  "daysToExpiry": 45,
  "estimatedUsageRate": 10,
  "estimatedDaysToStockout": 50,
  "willExpireBeforeStockout": true,
  "projectedWastage": 0,
  "recommendations": [
    "Consider promotional pricing",
    "Prioritize this batch for distribution",
    "Alert pharmacies about upcoming expiry"
  ]
}
```


Code Generation Prompt (Python Flask):

```
Create Flask endpoint for expiry risk prediction:
- Endpoint: POST /api/ai/predict/expiry-risk
- Calculate daysToExpiry (expiryDate - currentDate)
- Fetch historical usage rate for the medicine
- Calculate average daily usage from past 30 days
- Estimate days to stockout (currentStock / dailyUsage)
- Determine if medicine will expire before being used
- Calculate risk score based on:
  * Days to expiry (weight: 0.4)
  * Stock quantity (weight: 0.3)
  * Usage rate (weight: 0.3)
- Classify risk: LOW (< 0.3), MEDIUM (0.3-0.6), HIGH (0.6-0.8), CRITICAL (> 0.8)
- Project wastage quantity if stock won't be used
- Generate actionable recommendations based on risk level
- Return detailed analysis with suggestions
```

API 67: Get Stock Optimization Recommendations

Endpoint: `POST /api/ai/optimize/stock`
Request Body:


json

```
{
 "medicineIds": ["id1", "id2"],
 "optimizationGoal": "MINIMIZE_WASTAGE|MINIMIZE_STOCKOUT|BALANCE"
}
```

Response:


json

```
{
 "optimizationGoal": "BALANCE",
 "recommendations": [
  {
   "medicineId": "string",
   "medicineName": "string",
   "currentStock": 500,
   "currentReorderLevel": 100,
   "recommendedReorderLevel": 150,
   "recommendedOrderQuantity": 600,
   "reasoning": "Based on seasonal demand increase",
   "expectedBenefit": {
    "stockoutReduction": 25.0,
    "wastageReduction": 10.0,
    "costSaving": 5000.00
   }
  }
 ],
 "overallImpact": {
  "totalCostSaving": 50000.00,
  "wastageReduction": 15.0,
  "serviceImprovement": 20.0
 }
}
```

API 68: Detect Anomalies in Usage Pattern

Endpoint: `POST /api/ai/detect/anomalies`
Request Body:

json

```
{
  "medicineId": "string",
  "analysisWindow": 30
}
```

Response:

json

```
{
  "medicineId": "string",
  "medicineName": "string",
  "anomaliesDetected": true,
  "anomalyCount": 3,
  "anomalies": [
    {
      "date": "2025-10-15",
      "expectedUsage": 100,
      "actualUsage": 250,
      "deviationPercent": 150.0,
      "anomalyType": "SPIKE",
      "possibleCauses": [
        "Seasonal outbreak",
        "Bulk order from pharmacy",
        "Data entry error"
      ]
    }
  ],
  "recommendations": [
    "Investigate unusual spike on 2025-10-15",
    "Verify data accuracy",
    "Adjust forecasting model"
  ]
}
```

2.12 User & Profile Management APIs

API 69: Get Current User Profile

Endpoint: `GET /api/users/profile`
Headers: `Authorization: Bearer {token}`
Response:


json

```
{
  "id": "string",
  "fullName": "string",
  "email": "string",
  "phone": "string",
  "role": "ADMIN|SUPPLIER|PHARMACY",
  "organizationName": "string",
  "organizationId": "string",
  "licenseNumber": "string",
  "address": "string",
  "profilePicture": "url",
  "isEmailVerified": true,
  "isPhoneVerified": false,
  "preferences": {
    "language": "en|ne",
    "theme": "light|dark",
    "emailNotifications": true,
    "smsNotifications": false
  },
  "lastLogin": "datetime",
  "accountCreated": "datetime"
}
```

API 70: Update User Profile

Endpoint: `PUT /api/users/profile`
Request Body:


json

```
{
  "fullName": "string",
  "phone": "string",
  "address": "string",
  "organizationName": "string"
}
```

API 71: Upload Profile Picture

Endpoint: `POST /api/users/profile/picture`
Request: Multipart file
Response:


json

```
{
  "success": true,
  "message": "Profile picture updated",
  "imageUrl": "string"
}
```

Code Generation Prompt:

```
Create Spring Boot POST endpoint for profile picture upload:
- Endpoint: POST /api/users/profile/picture
- Accept multipart/form-data with image file
- Validate file type (jpg, png, max 5MB)
- Resize image to 400x400 pixels using ImageIO
- Upload to cloud storage (AWS S3 or local storage)
- Generate unique filename (userId_timestamp.jpg)
- Update user's profilePicture field in database
- Delete old profile picture if exists
- Return new image URL
- Handle upload failures gracefully
```

API 72: Change Password

Endpoint: `POST /api/users/change-password`
Request Body:

json

```
{
  "currentPassword": "string",
  "newPassword": "string",
  "confirmPassword": "string"
}
```

Code Generation Prompt:

```
Create Spring Boot POST endpoint for password change:
- Endpoint: POST /api/users/change-password
- Verify currentPassword matches stored hash
- Validate newPassword meets requirements:
  * Minimum 8 characters
  * At least one uppercase, lowercase, number
  * At least one special character
- Verify newPassword matches confirmPassword
- Hash newPassword using BCrypt
- Update user's password in database
- Invalidate all existing tokens/sessions
- Send email notification about password change
- Return success response
- Log password change activity
```

API 73: Update Notification Preferences

Endpoint: `PUT /api/users/preferences/notifications`
Request Body:

json

```
{
  "emailNotifications": true,
  "smsNotifications": false,
  "lowStockAlerts": true,
  "expiryAlerts": true,
  "orderAlerts": true,
  "systemAlerts": true
}
```


API 74: Get All Users (Admin Only)

Endpoint: `GET /api/users?page=0&size=10&role=&status=&search=`
Response:


json


```
{
  "content": [
    {
      "id": "string",
      "fullName": "string",
      "email": "string",
      "phone": "string",
      "role": "string",
      "organizationName": "string",
      "status": "ACTIVE|INACTIVE|SUSPENDED",
      "lastLogin": "datetime",
      "accountCreated": "datetime"
    }
  ],
  "totalElements": 50
}
```


API 75: Deactivate User (Admin Only)

Endpoint: `PUT /api/users/{id}/deactivate`
Request Body:


json

```
{
  "reason": "string"
}
```

API 76: Activate User (Admin Only)

Endpoint: `PUT /api/users/{id}/activate`

API 77: Reset User Password (Admin Only)

Endpoint: `POST /api/users/{id}/reset-password`
Response:


json

```
{
  "success": true,
  "message": "Password reset link sent to user",
  "temporaryPassword": "string"
}
```

2.13 Activity Log & Audit APIs

API 78: Get Activity Logs

Endpoint: `GET /api/logs/activities?page=0&size=50&startDate=&endDate=&userId=&activityType=`
Response:


json

```
{
 "content": [
  {
   "id": "string",
   "timestamp": "datetime",
   "userId": "string",
   "userName": "string",
   "userRole": "string",
   "activityType": "LOGIN|LOGOUT|MEDICINE_ADDED|STOCK_ADJUSTED|DISTRIBUTION_CREATED|etc"
   "entityType": "MEDICINE|BATCH|DISTRIBUTION|USER",
   "entityId": "string",
   "description": "string",
   "ipAddress": "string",
   "userAgent": "string",
   "status": "SUCCESS|FAILED"
  }
 ],
 "totalElements": 500
}
```

Code Generation Prompt:

```
Create Spring Boot GET endpoint for activity logs:
- Endpoint: GET /api/logs/activities
- Support multiple filters (date range, user, activity type)
- Fetch from activity_logs table with JOIN on users table
- Sort by timestamp descending (most recent first)
- Include pagination
- Only ADMIN can access all logs
- Other roles can only see their own activity logs
- Return detailed log information including IP and user agent
```

API 79: Get Audit Trail for Entity

Endpoint: `GET /api/audit/{entityType}/{entityId}`
Response:

```
json
```

```
{
  "entityType": "MEDICINE",
  "entityId": "string",
  "entityName": "Paracetamol 500mg",
  "history": [
    {
      "timestamp": "datetime",
      "action": "CREATED|UPDATED|DELETED",
      "performedBy": "string",
      "changes": [
        {
          "field": "unitPrice",
          "oldValue": "10.00",
          "newValue": "12.00"
        }
      ],
      "notes": "string"
    }
  ]
}
```


API 80: Log Custom Activity

Endpoint: `POST /api/logs/activity`
Request Body:


json


```
{
  "activityType": "CUSTOM",
  "description": "string",
  "entityType": "string",
  "entityId": "string"
}
```

2.14 System Configuration APIs

API 81: Get System Settings

Endpoint: `GET /api/settings`
Response:

json


```
{
 "general": {
  "systemName": "MediTrack",
  "contactEmail": "admin@meditrack.com",
  "contactPhone": "+977-XXXXXXXXXX",
  "address": "string",
  "supportedLanguages": ["en", "ne"]
 },
 "notifications": {
  "emailEnabled": true,
  "smsEnabled": false,
  "lowStockThreshold": 20,
  "expiryWarningDays": 90,
  "criticalExpiryDays": 30
 },
 "stock": {
  "defaultReorderLevel": 100,
  "autoGeneratePO": false,
  "stockAdjustmentApproval": true
 },
 "security": {
  "passwordMinLength": 8,
  "sessionTimeout": 30,
  "maxLoginAttempts": 5,
  "twoFactorEnabled": false
 },
 "backup": {
  "autoBackupEnabled": true,
  "backupFrequency": "DAILY",
  "lastBackupDate": "datetime",
  "backupRetentionDays": 30
 }
}
```


API 82: Update System Settings (Admin Only)

Endpoint: `PUT /api/settings`
Request Body: (same structure as GET response)

API 83: Test Email Configuration

Endpoint: `POST /api/settings/test-email`
Request Body:


json

```
{
  "recipientEmail": "string"
}
```

Response:


json

```
{
  "success": true,
  "message": "Test email sent successfully"
}
```

API 84: Trigger Manual Backup

Endpoint: `POST /api/settings/backup`
Response:


json

```
{
  "success": true,
  "message": "Backup initiated",
  "backupId": "string",
  "estimatedTime": "5 minutes"
}
```

API 85: Get System Health Status

Endpoint: `GET /api/settings/health`
Response:


json

```
{
 "status": "HEALTHY|DEGRADED|CRITICAL",
 "timestamp": "datetime",
 "components": {
  "database": {
   "status": "UP",
   "responseTime": 15
  },
  "storage": {
   "status": "UP",
   "usedSpace": "45%"
  },
  "aiService": {
   "status": "UP",
   "lastCheck": "datetime"
  },
  "emailService": {
   "status": "UP"
  }
 },
 "metrics": {
  "totalUsers": 100,
  "activeUsers": 45,
  "todayTransactions": 150,
  "averageResponseTime": 250
 }
}
```

2.15 Search & Filter APIs

API 86: Global Search

Endpoint: `GET /api/search?query=&type=ALL|MEDICINE|SUPPLIER|PHARMACY|BATCH`
Response:

json

```
{
 "query": "paracetamol",
 "totalResults": 25,
 "results": {
  "medicines": [
   {
    "id": "string",
    "name": "Paracetamol 500mg",
    "type": "MEDICINE",
    "category": "Painkiller",
    "stock": 500
   }
  ],
  "batches": [
   {
    "id": "string",
    "batchNumber": "string",
    "type": "BATCH",
    "medicineName": "Paracetamol 500mg",
    "expiryDate": "date"
   }
  ],
  "suppliers": [],
  "pharmacies": []
 }
}
```


Code Generation Prompt:

```
Create Spring Boot GET endpoint for global search:
- Endpoint: GET /api/search
- Accept query string and optional type filter
- Search across multiple tables:
  * Medicines: name, genericName, manufacturer
  * Batches: batchNumber
  * Suppliers: companyName, contactPerson
  * Pharmacies: pharmacyName, ownerName
- Use LIKE queries with wildcards (%query%)
- Return results grouped by type
- Limit results per type (max 10 each)
- Sort by relevance (exact matches first)
- Respect role-based access (filter results by user role)
- Return empty arrays for types with no matches
```

API 87: Advanced Medicine Filter

Endpoint: `POST /api/medicines/filter`
Request Body:

json

```
{
  "categories": ["Antibiotic", "Painkiller"],
  "suppliers": ["supplier1", "supplier2"],
  "stockStatus": ["LOW_STOCK", "OUT_OF_STOCK"],
  "priceRange": {
    "min": 10.00,
    "max": 100.00
  },
  "expiryDateRange": {
    "from": "date",
    "to": "date"
  },
  "page": 0,
  "size": 20,
  "sortBy": "name",
  "sortOrder": "ASC"
}
```

Response: Paginated list of medicines matching all criteria

2.16 Export & Import APIs

API 88: Export Medicines Data

Endpoint: `GET /api/export/medicines?format=CSV|EXCEL|JSON`
Response: File download

API 89: Import Medicines Data

Endpoint: `POST /api/import/medicines`
Request: Multipart file (CSV/Excel)
Response:

json


```
{
  "success": true,
  "processed": 100,
  "imported": 95,
  "failed": 5,
  "errors": [
    {
      "row": 15,
      "field": "category",
      "error": "Invalid category name",
      "data": {...}
    }
  ]
}
```


API 90: Export Stock Report

Endpoint: `GET /api/export/stock-report?format=PDF|EXCEL&asOf=date`

API 91: Export Distribution History

Endpoint: `GET /api/export/distributions?startDate=&endDate=&format=EXCEL|CSV`

2.17 Barcode/QR Code APIs

API 92: Generate Barcode for Medicine

Endpoint: `POST /api/barcode/generate`
Request Body:

json

```
{
  "medicineId": "string",
  "batchNumber": "string",
  "format": "EAN13|QR"
}
```

Response:


json

```
{
  "success": true,
  "barcodeData": "string",
  "barcodeImage": "base64_encoded_image",
  "downloadUrl": "string"
}
```

Code Generation Prompt:

```
Create Spring Boot POST endpoint for barcode generation:
- Endpoint: POST /api/barcode/generate
- Use ZXing library for barcode/QR code generation
- For EAN13: encode medicine ID and batch number
- For QR: encode JSON with full medicine and batch details
- Generate barcode image (PNG format, 300x150 pixels)
- Convert image to base64 for inline display
- Save image to storage and return download URL
- Return both image data and URL
- Only ADMIN and SUPPLIER can generate barcodes
```

API 93: Scan and Decode Barcode

Endpoint: `POST /api/barcode/scan`
Request Body:


json

```
{
  "barcodeImage": "base64_encoded_image"
}
```

Response:


json

```
{
  "success": true,
  "decoded": true,
  "data": {
    "medicineId": "string",
    "medicineName": "string",
    "batchNumber": "string",
    "expiryDate": "date",
    "currentStock": 0
  }
}
```

2.18 Integration APIs

API 94: Webhook for Stock Updates

Endpoint: `POST /api/webhooks/stock-update`
Request Body:


json

```
{
  "medicineId": "string",
  "eventType": "STOCK_LOW|STOCK_OUT|STOCK_REPLENISHED",
  "currentStock": 0,
  "reorderLevel": 0,
  "timestamp": "datetime"
}
```

API 95: API Key Management (For third-party integrations)

Endpoint: `POST /api/integrations/api-keys`
Request Body:


json

```
{
  "name": "string",
  "description": "string",
  "permissions": ["READ_MEDICINES", "READ_STOCK"],
  "expiresAt": "date"
}
```

Response:


json

```
{
  "success": true,
  "apiKey": "generated_api_key",
  "message": "API key created successfully"
}
```

---

3. DATABASE SCHEMA REQUIREMENTS

3.1 Core Tables

Table: users


sql

```sql
CREATE TABLE users (
  id VARCHAR(36) PRIMARY KEY,
  full_name VARCHAR(100) NOT NULL,
  email VARCHAR(100) UNIQUE NOT NULL,
  password_hash VARCHAR(255) NOT NULL,
  phone VARCHAR(20),
  role ENUM('ADMIN', 'SUPPLIER', 'PHARMACY') NOT NULL,
  organization_id VARCHAR(36),
  organization_name VARCHAR(200),
  license_number VARCHAR(50),
  address TEXT,
  profile_picture VARCHAR(255),
  is_email_verified BOOLEAN DEFAULT FALSE,
  is_phone_verified BOOLEAN DEFAULT FALSE,
  status ENUM('ACTIVE', 'INACTIVE', 'SUSPENDED') DEFAULT 'ACTIVE',
  language ENUM('en', 'ne') DEFAULT 'en',
  theme ENUM('light', 'dark') DEFAULT 'light',
  email_notifications BOOLEAN DEFAULT TRUE,
  sms_notifications BOOLEAN DEFAULT FALSE,
  last_login TIMESTAMP NULL,
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
  INDEX idx_email (email),
  INDEX idx_role (role),
  INDEX idx_organization_id (organization_id)
);
```

Table: medicines

sql

```sql
CREATE TABLE medicines (
  id VARCHAR(36) PRIMARY KEY,
  name VARCHAR(200) NOT NULL,
  generic_name VARCHAR(200),
  category VARCHAR(100) NOT NULL,
  manufacturer VARCHAR(200),
  description TEXT,
  unit_price DECIMAL(10, 2) NOT NULL,
  current_stock INT DEFAULT 0,
  reorder_level INT DEFAULT 100,
  image_url VARCHAR(255),
  status ENUM('AVAILABLE', 'LOW_STOCK', 'OUT_OF_STOCK') DEFAULT 'AVAILABLE',
  is_active BOOLEAN DEFAULT TRUE,
  created_by VARCHAR(36),
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
  INDEX idx_name (name),
  INDEX idx_category (category),
  INDEX idx_status (status),
  FOREIGN KEY (created_by) REFERENCES users(id)
);
```

Table: batches

sql

```sql
CREATE TABLE batches (
  id VARCHAR(36) PRIMARY KEY,
  batch_number VARCHAR(100) UNIQUE NOT NULL,
  medicine_id VARCHAR(36) NOT NULL,
  supplier_id VARCHAR(36) NOT NULL,
  quantity INT NOT NULL,
  manufacturing_date DATE NOT NULL,
  expiry_date DATE NOT NULL,
  purchase_price DECIMAL(10, 2) NOT NULL,
  storage_location VARCHAR(100),
  notes TEXT,
  status ENUM('ACTIVE', 'NEAR_EXPIRY', 'EXPIRED') DEFAULT 'ACTIVE',
  is_active BOOLEAN DEFAULT TRUE,
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
  INDEX idx_batch_number (batch_number),
  INDEX idx_medicine_id (medicine_id),
  INDEX idx_supplier_id (supplier_id),
  INDEX idx_expiry_date (expiry_date),
  FOREIGN KEY (medicine_id) REFERENCES medicines(id),
  FOREIGN KEY (supplier_id) REFERENCES suppliers(id)
);
```

Table: suppliers


sql

```sql
CREATE TABLE suppliers (
  id VARCHAR(36) PRIMARY KEY,
  company_name VARCHAR(200) NOT NULL,
  contact_person VARCHAR(100) NOT NULL,
  email VARCHAR(100) UNIQUE NOT NULL,
  phone VARCHAR(20) NOT NULL,
  address TEXT,
  license_number VARCHAR(50) UNIQUE NOT NULL,
  payment_terms VARCHAR(200),
  bank_account_name VARCHAR(100),
  bank_account_number VARCHAR(50),
  bank_name VARCHAR(100),
  notes TEXT,
  rating DECIMAL(3, 2) DEFAULT 0.00,
  total_supplies INT DEFAULT 0,
  status ENUM('ACTIVE', 'INACTIVE') DEFAULT 'ACTIVE',
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
  INDEX idx_company_name (company_name),
  INDEX idx_status (status)
);
```

Table: supplier_medicines (Many-to-Many)


sql

```sql
CREATE TABLE supplier_medicines (
  id VARCHAR(36) PRIMARY KEY,
  supplier_id VARCHAR(36) NOT NULL,
  medicine_id VARCHAR(36) NOT NULL,
  unit_price DECIMAL(10, 2),
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  UNIQUE KEY unique_supplier_medicine (supplier_id, medicine_id),
  FOREIGN KEY (supplier_id) REFERENCES suppliers(id) ON DELETE CASCADE,
  FOREIGN KEY (medicine_id) REFERENCES medicines(id) ON DELETE CASCADE
);
```

Table: pharmacies


sql

```sql
CREATE TABLE pharmacies (
  id VARCHAR(36) PRIMARY KEY,
  pharmacy_name VARCHAR(200) NOT NULL,
  owner_name VARCHAR(100) NOT NULL,
  manager_name VARCHAR(100),
  email VARCHAR(100) UNIQUE NOT NULL,
  phone VARCHAR(20) NOT NULL,
  address TEXT NOT NULL,
  license_number VARCHAR(50) UNIQUE NOT NULL,
  operating_hours VARCHAR(100),
  status ENUM('ACTIVE', 'INACTIVE') DEFAULT 'ACTIVE',
  total_orders INT DEFAULT 0,
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
  INDEX idx_pharmacy_name (pharmacy_name),
  INDEX idx_status (status)
);
```

Table: distributions

sql

```sql
CREATE TABLE distributions (
  id VARCHAR(36) PRIMARY KEY,
  distribution_number VARCHAR(50) UNIQUE NOT NULL,
  pharmacy_id VARCHAR(36) NOT NULL,
  distribution_date DATE NOT NULL,
  expected_delivery_date DATE NOT NULL,
  actual_delivery_date DATE NULL,
  status ENUM('PENDING', 'IN_TRANSIT', 'DELIVERED', 'CANCELLED') DEFAULT 'PENDING',
  priority ENUM('NORMAL', 'URGENT') DEFAULT 'NORMAL',
  total_value DECIMAL(12, 2) NOT NULL,
  tracking_number VARCHAR(100),
  shipping_address TEXT,
  notes TEXT,
  created_by VARCHAR(36) NOT NULL,
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
  INDEX idx_distribution_number (distribution_number),
  INDEX idx_pharmacy_id (pharmacy_id),
  INDEX idx_status (status),
  INDEX idx_distribution_date (distribution_date),
  FOREIGN KEY (pharmacy_id) REFERENCES pharmacies(id),
  FOREIGN KEY (created_by) REFERENCES users(id)
);
```

Table: distribution_items


sql

```sql
CREATE TABLE distribution_items (
  id VARCHAR(36) PRIMARY KEY,
  distribution_id VARCHAR(36) NOT NULL,
  medicine_id VARCHAR(36) NOT NULL,
  batch_number VARCHAR(100) NOT NULL,
  quantity INT NOT NULL,
  unit_price DECIMAL(10, 2) NOT NULL,
  total_price DECIMAL(12, 2) NOT NULL,
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  FOREIGN KEY (distribution_id) REFERENCES distributions(id) ON DELETE CASCADE,
  FOREIGN KEY (medicine_id) REFERENCES medicines(id),
  INDEX idx_distribution_id (distribution_id)
);
```

Table: purchase_orders


sql

```sql
CREATE TABLE purchase_orders (
  id VARCHAR(36) PRIMARY KEY,
  po_number VARCHAR(50) UNIQUE NOT NULL,
  supplier_id VARCHAR(36) NOT NULL,
  order_date DATE NOT NULL,
  expected_delivery_date DATE NOT NULL,
  actual_delivery_date DATE NULL,
  subtotal DECIMAL(12, 2) NOT NULL,
  tax DECIMAL(10, 2) DEFAULT 0.00,
  total_amount DECIMAL(12, 2) NOT NULL,
  status ENUM('DRAFT', 'SENT', 'CONFIRMED', 'RECEIVED', 'CANCELLED') DEFAULT 'DRAFT',
  payment_terms VARCHAR(200),
  invoice_number VARCHAR(100),
  invoice_amount DECIMAL(12, 2),
  notes TEXT,
  created_by VARCHAR(36) NOT NULL,
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
  INDEX idx_po_number (po_number),
  INDEX idx_supplier_id (supplier_id),
  INDEX idx_status (status),
  FOREIGN KEY (supplier_id) REFERENCES suppliers(id),
  FOREIGN KEY (created_by) REFERENCES users(id)
);
```

Table: purchase_order_items

sql

```sql
CREATE TABLE purchase_order_items (
  id VARCHAR(36) PRIMARY KEY,
  po_id VARCHAR(36) NOT NULL,
  medicine_id VARCHAR(36) NOT NULL,
  quantity INT NOT NULL,
  unit_price DECIMAL(10, 2) NOT NULL,
  total_price DECIMAL(12, 2) NOT NULL,
  received_quantity INT DEFAULT 0,
  damage_quantity INT DEFAULT 0,
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  FOREIGN KEY (po_id) REFERENCES purchase_orders(id) ON DELETE CASCADE,
  FOREIGN KEY (medicine_id) REFERENCES medicines(id),
  INDEX idx_po_id (po_id)
);
```

Table: stock_transactions

sql

```sql
CREATE TABLE stock_transactions (
  id VARCHAR(36) PRIMARY KEY,
  medicine_id VARCHAR(36) NOT NULL,
  batch_number VARCHAR(100),
  transaction_type ENUM('ADD', 'REMOVE', 'TRANSFER', 'ADJUSTMENT') NOT NULL,
  quantity INT NOT NULL,
  reason VARCHAR(100) NOT NULL,
  reference_type ENUM('PURCHASE_ORDER', 'DISTRIBUTION', 'MANUAL', 'EXPIRY', 'DAMAGE'),
  reference_id VARCHAR(36),
  from_location VARCHAR(100),
  to_location VARCHAR(100),
  notes TEXT,
  performed_by VARCHAR(36) NOT NULL,
  transaction_date DATE NOT NULL,
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  INDEX idx_medicine_id (medicine_id),
  INDEX idx_transaction_date (transaction_date),
  FOREIGN KEY (medicine_id) REFERENCES medicines(id),
  FOREIGN KEY (performed_by) REFERENCES users(id)
);
```

Table: notifications

sql

```sql
CREATE TABLE notifications (
  id VARCHAR(36) PRIMARY KEY,
  user_id VARCHAR(36) NOT NULL,
  type ENUM('LOW_STOCK', 'EXPIRY', 'ORDER', 'SYSTEM', 'CUSTOM') NOT NULL,
  title VARCHAR(200) NOT NULL,
  message TEXT NOT NULL,
  priority ENUM('LOW', 'MEDIUM', 'HIGH', 'CRITICAL') DEFAULT 'MEDIUM',
  related_entity_type VARCHAR(50),
  related_entity_id VARCHAR(36),
  action_url VARCHAR(255),
  is_read BOOLEAN DEFAULT FALSE,
  read_at TIMESTAMP NULL,
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  INDEX idx_user_id (user_id),
  INDEX idx_is_read (is_read),
  INDEX idx_created_at (created_at),
  FOREIGN KEY (user_id) REFERENCES users(id) ON DELETE CASCADE
);
```

Table: activity_logs

sql

```sql
CREATE TABLE activity_logs (
  id VARCHAR(36) PRIMARY KEY,
  user_id VARCHAR(36),
  activity_type VARCHAR(100) NOT NULL,
  entity_type VARCHAR(50),
  entity_id VARCHAR(36),
  description TEXT NOT NULL,
  ip_address VARCHAR(45),
  user_agent TEXT,
  status ENUM('SUCCESS', 'FAILED') DEFAULT 'SUCCESS',
  error_message TEXT,
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  INDEX idx_user_id (user_id),
  INDEX idx_activity_type (activity_type),
  INDEX idx_created_at (created_at),
  FOREIGN KEY (user_id) REFERENCES users(id) ON DELETE SET NULL
);
```

Table: audit_trail

sql

```sql
CREATE TABLE audit_trail (
  id VARCHAR(36) PRIMARY KEY,
  entity_type VARCHAR(50) NOT NULL,
  entity_id VARCHAR(36) NOT NULL,
  action ENUM('CREATED', 'UPDATED', 'DELETED') NOT NULL,
  field_name VARCHAR(100),
  old_value TEXT,
  new_value TEXT,
  performed_by VARCHAR(36) NOT NULL,
  notes TEXT,
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  INDEX idx_entity (entity_type, entity_id),
  INDEX idx_created_at (created_at),
  FOREIGN KEY (performed_by) REFERENCES users(id)
);
```

Table: supplier_ratings

sql

```sql
CREATE TABLE supplier_ratings (
  id VARCHAR(36) PRIMARY KEY,
  supplier_id VARCHAR(36) NOT NULL,
  po_id VARCHAR(36),
  rating DECIMAL(3, 2) NOT NULL CHECK (rating >= 0 AND rating <= 5),
  comment TEXT,
  rated_by VARCHAR(36) NOT NULL,
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  FOREIGN KEY (supplier_id) REFERENCES suppliers(id) ON DELETE CASCADE,
  FOREIGN KEY (po_id) REFERENCES purchase_orders(id),
  FOREIGN KEY (rated_by) REFERENCES users(id)
);
```

Table: system_settings

sql

```sql
CREATE TABLE system_settings (
  id VARCHAR(36) PRIMARY KEY,
  setting_key VARCHAR(100) UNIQUE NOT NULL,
  setting_value TEXT NOT NULL,
  setting_type ENUM('STRING', 'NUMBER', 'BOOLEAN', 'JSON') DEFAULT 'STRING',
  category VARCHAR(50) NOT NULL,
  description TEXT,
  updated_by VARCHAR(36),
  updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
  FOREIGN KEY (updated_by) REFERENCES users(id)
);
```

Table: distribution_timeline

sql

```sql
CREATE TABLE distribution_timeline (
  id VARCHAR(36) PRIMARY KEY,
  distribution_id VARCHAR(36) NOT NULL,
  status VARCHAR(50) NOT NULL,
  note TEXT,
  timestamp TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  updated_by VARCHAR(36),
  FOREIGN KEY (distribution_id) REFERENCES distributions(id) ON DELETE CASCADE,
  FOREIGN KEY (updated_by) REFERENCES users(id),
  INDEX idx_distribution_id (distribution_id)
);
```

Table: po_timeline


sql

```sql
CREATE TABLE po_timeline (
  id VARCHAR(36) PRIMARY KEY,
  po_id VARCHAR(36) NOT NULL,
  status VARCHAR(50) NOT NULL,
  note TEXT,
  timestamp TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  updated_by VARCHAR(36),
  FOREIGN KEY (po_id) REFERENCES purchase_orders(id) ON DELETE CASCADE,
  FOREIGN KEY (updated_by) REFERENCES users(id),
  INDEX idx_po_id (po_id)
);
```

---

4. SECURITY REQUIREMENTS

4.1 Authentication & Authorization

Spring Security Configuration Prompt:

Create Spring Security configuration for MediTrack:
- Use Spring Security 6.x
- Implement JWT-based authentication
- Configure stateless session management
- Define role hierarchy: ADMIN > SUPPLIER, ADMIN > PHARMACY
- Configure CORS to allow React frontend (localhost:3000 and production domain)
- Implement custom UserDetailsService loading users from database
- Create JwtAuthenticationFilter to validate tokens on each request
- Configure password encoder using BCryptPasswordEncoder (strength: 12)
- Implement login attempt limiting (max 5 attempts, lock for 15 minutes)
- Add security headers (X-Content-Type-Options, X-Frame-Options, etc.)
- Configure CSRF protection for non-API endpoints
- Implement refresh token rotation

4.2 Role-Based Access Control

Permission Matrix:

ADMIN:
- Full access to all modules
- User management
- System settings
- All reports
- Approve/reject operations

SUPPLIER:
- View assigned products
- View and confirm purchase orders
- Update delivery status
- Upload invoices
- View own performance metrics

PHARMACY:
- View inventory
- Request distributions
- Confirm receipt of medicines
- View own orders and history
- Report issues

4.3 Data Protection

- All passwords hashed with BCrypt (cost factor: 12)
- Sensitive fields encrypted at rest (bank details, license numbers)
- JWT tokens expire after 1 hour
- Refresh tokens expire after 7 days
- SSL/TLS for all API communications
- Input validation on all endpoints
- SQL injection prevention using JPA/Hibernate
- XSS protection with content security policies

---

5. FRONTEND COMPONENT STRUCTURE

5.1 Component Hierarchy

```
src/
├── components/
│   ├── common/
│   │   ├── Navbar.jsx
│   │   ├── Sidebar.jsx
│   │   ├── Footer.jsx
│   │   ├── LoadingSpinner.jsx
│   │   ├── ErrorBoundary.jsx
│   │   ├── ConfirmDialog.jsx
│   │   ├── Toast.jsx
│   │   └── SearchBar.jsx
│   ├── auth/
│   │   ├── LoginForm.jsx
│   │   ├── RegisterForm.jsx
│   │   ├── ForgotPasswordForm.jsx
│   │   └── ProtectedRoute.jsx
│   ├── dashboard/
│   │   ├── AdminDashboard.jsx
│   │   ├── SupplierDashboard.jsx
│   │   ├── PharmacyDashboard.jsx
│   │   ├── StatCard.jsx
│   │   ├── ActivityTimeline.jsx
│   │   └── ChartContainer.jsx
│   ├── medicine/
│   │   ├── MedicineList.jsx
│   │   ├── MedicineCard.jsx
│   │   ├── MedicineForm.jsx
│   │   ├── MedicineDetails.jsx
│   │   └── MedicineFilters.jsx
│   ├── batch/
│   │   ├── BatchList.jsx
│   │   ├── BatchForm.jsx
│   │   ├── ExpiryAlerts.jsx
│   │   └── BatchTimeline.jsx
│   ├── stock/
│   │   ├── StockOverview.jsx
│   │   ├── StockAdjustmentForm.jsx
│   │   ├── LowStockAlerts.jsx
│   │   └── StockHistory.jsx
│   ├── supplier/
│   │   ├── SupplierList.jsx
│   │   ├── SupplierForm.jsx
│   │   ├── SupplierDetails.jsx
│   │   └── SupplierPerformance.jsx
```

```
│   ├── pharmacy/
│   │   ├── PharmacyList.jsx
│   │   ├── PharmacyForm.jsx
│   │   ├── PharmacyDetails.jsx
│   │   └── PharmacyInventory.jsx
│   ├── distribution/
│   │   ├── DistributionList.jsx
│   │   ├── DistributionForm.jsx
│   │   ├── DistributionDetails.jsx
│   │   └── DistributionTracking.jsx
│   ├── purchase/
│   │   ├── PurchaseOrderList.jsx
│   │   ├── PurchaseOrderForm.jsx
│   │   ├── PurchaseOrderDetails.jsx
│   │   └── ReceiveOrder.jsx
│   ├── reports/
│   │   ├── ReportDashboard.jsx
│   │   ├── StockReport.jsx
│   │   ├── DistributionReport.jsx
│   │   ├── ExpiryReport.jsx
│   │   └── ReportFilters.jsx
│   ├── notifications/
│   │   ├── NotificationCenter.jsx
│   │   ├── NotificationItem.jsx
│   │   └── NotificationBell.jsx
│   ├── profile/
│   │   ├── ProfileView.jsx
│   │   ├── ProfileEdit.jsx
│   │   ├── ChangePassword.jsx
│   │   └── NotificationSettings.jsx
│   └── admin/
│       ├── UserManagement.jsx
│       ├── SystemSettings.jsx
│       ├── ActivityLogs.jsx
│       └── SystemHealth.jsx
├── pages/
│   ├── LandingPage.jsx
│   ├── LoginPage.jsx
│   ├── RegisterPage.jsx
│   ├── DashboardPage.jsx
│   ├── MedicinesPage.jsx
│   ├── BatchesPage.jsx
│   ├── StockPage.jsx
│   ├── SuppliersPage.jsx
```

```
│   ├── PharmaciesPage.jsx
│   ├── DistributionsPage.jsx
│   ├── PurchaseOrdersPage.jsx
│   ├── ReportsPage.jsx
│   ├── NotificationsPage.jsx
│   ├── ProfilePage.jsx
│   ├── SettingsPage.jsx
│   └── NotFoundPage.jsx
├── services/
│   ├── api.js (Axios configuration)
│   ├── authService.js
│   ├── medicineService.js
│   ├── batchService.js
│   ├── stockService.js
│   ├── supplierService.js
│   ├── pharmacyService.js
│   ├── distributionService.js
│   ├── purchaseOrderService.js
│   ├── reportService.js
│   └── notificationService.js
├── context/
│   ├── AuthContext.jsx
│   ├── ThemeContext.jsx
│   └── NotificationContext.jsx
├── hooks/
│   ├── useAuth.js
│   ├── useApi.js
│   ├── useDebounce.js
│   └── usePagination.js
├── utils/
│   ├── constants.js
│   ├── validators.js
│   ├── formatters.js
│   └── helpers.js
└── App.jsx
```

---

6. CODE GENERATION PROMPTS

6.1 Backend Setup Prompt

Create a Spring Boot project for MediTrack medicine management system with the following specifications:

PROJECT SETUP:
- Spring Boot version: 3.2.x
- Java version: 17
- Build tool: Maven
- Package structure: com.meditrack

DEPENDENCIES:
- spring-boot-starter-web (REST API)
- spring-boot-starter-data-jpa (Database access)
- spring-boot-starter-security (Authentication)
- spring-boot-starter-validation (Input validation)
- spring-boot-starter-mail (Email notifications)
- mysql-connector-j (MySQL driver)
- jjwt-api, jjwt-impl, jjwt-jackson (JWT authentication)
- lombok (Reduce boilerplate code)
- springdoc-openapi-starter-webmvc-ui (API documentation)
- modelmapper (DTO mapping)

APPLICATION STRUCTURE:
1. Create entities for: User, Medicine, Batch, Supplier, Pharmacy, Distribution, PurchaseOrder
2. Create repositories extending JpaRepository for each entity
3. Create DTOs for request/response objects
4. Create service layer with business logic
5. Create REST controllers with proper endpoints
6. Implement JWT authentication with Spring Security
7. Configure MySQL connection in application.properties
8. Add global exception handler
9. Configure CORS for React frontend
10. Add Swagger/OpenAPI documentation

CONFIGURATION (application.properties):
- Database: meditrack_db
- Server port: 8080
- JWT secret key and expiration
- Email SMTP settings
- File upload directory
- Logging configuration

Create the complete project structure with all necessary files and configurations.

6.2 Frontend Setup Prompt

Create a React application for MediTrack medicine management system with the following specifications:

PROJECT SETUP:
- React version: 18.x
- Node version: 18+
- Package manager: npm
- Build tool: Vite

DEPENDENCIES:
- react-router-dom (Routing)
- axios (HTTP client)
- tailwindcss (Styling)
- chart.js & react-chartjs-2 (Charts)
- react-hook-form (Form handling)
- yup (Form validation)
- react-toastify (Notifications)
- date-fns (Date formatting)
- lucide-react (Icons)
- zustand or context API (State management)

PROJECT STRUCTURE:
1. Set up routing with React Router
2. Create authentication context
3. Set up Axios interceptors for JWT token
4. Create reusable components (navbar, sidebar, forms, tables)
5. Implement role-based routing and component rendering
6. Create service layer for API calls
7. Set up Tailwind CSS with custom theme
8. Implement responsive design
9. Add loading states and error boundaries
10. Configure environment variables

FEATURES TO IMPLEMENT:
- Login/Register forms with validation
- Dashboard with charts and statistics
- Medicine CRUD operations with search/filter
- Batch management with expiry alerts
- Distribution tracking
- Purchase order management
- Reports generation
- Notification system
- User profile management

Create the complete folder structure with all necessary components, pages, and utilities.

6.3 AI Module Setup Prompt (Python Flask)

Create a Python Flask microservice for AI-powered demand forecasting and expiry prediction for MediTrack:

PROJECT SETUP:
- Python version: 3.9+
- Framework: Flask
- Environment: Virtual environment (venv)

DEPENDENCIES:
- flask (Web framework)
- flask-cors (CORS handling)
- pandas (Data manipulation)
- numpy (Numerical operations)
- scikit-learn (Machine learning)
- tensorflow or pytorch (Deep learning)
- prophet (Time series forecasting)
- mysql-connector-python (Database connection)
- python-dotenv (Environment variables)

PROJECT STRUCTURE:
/ai-service/
├── app.py (Main Flask application)
├── models/
│   ├── demand_forecaster.py
│   ├── expiry_predictor.py
│   └── stock_optimizer.py
├── utils/
│   ├── data_preprocessor.py
│   ├── db_connector.py
│   └── validators.py
├── config.py
├── requirements.txt
└── .env

FEATURES TO IMPLEMENT:
1. Demand Forecasting API:
   - Accept medicine ID and forecast period
   - Fetch historical distribution data
   - Train time series model (ARIMA/Prophet/LSTM)
   - Generate predictions with confidence intervals
   - Return forecast with recommendations

2. Expiry Risk Prediction API:
   - Calculate expiry risk based on multiple factors
   - Predict if medicine will be used before expiry

- Estimate wastage quantity
- Provide actionable recommendations

3. Stock Optimization API:
- Analyze historical usage patterns
- Recommend optimal reorder levels
- Calculate safety stock
- Minimize wastage while preventing stockouts

4. Anomaly Detection API:
- Detect unusual usage patterns
- Identify spikes or drops in demand
- Alert on data inconsistencies

CONFIGURATION:
- Database connection to MySQL
- Model persistence (save/load trained models)
- API endpoints with proper error handling
- CORS configuration for Spring Boot backend
- Logging configuration

Create the complete Flask application with all ML models and endpoints.

---

7. TESTING REQUIREMENTS

7.1 Backend Testing

Unit Testing Prompt:

Create JUnit 5 test cases for MediTrack backend:

1. Repository Tests:
   - Test CRUD operations for all entities
   - Test custom query methods
   - Test relationships and cascading

2. Service Tests:
   - Mock repository dependencies
   - Test business logic
   - Test exception handling
   - Test validation logic

3. Controller Tests:
   - Use MockMvc for endpoint testing
   - Test request/response mappings
   - Test authentication and authorization
   - Test error responses

4. Integration Tests:
   - Test end-to-end workflows
   - Test with embedded H2 database
   - Test API endpoints with real database

Test Coverage Target: Minimum 80%

7.2 Frontend Testing

Component Testing Prompt:

Create Jest and React Testing Library tests for MediTrack frontend:

1. Component Tests:
   - Test rendering and props
   - Test user interactions
   - Test conditional rendering
   - Test form submissions

2. Integration Tests:
   - Test API service calls with mock data
   - Test routing and navigation
   - Test authentication flow

3. E2E Tests (Optional with Cypress):
   - Test complete user workflows
   - Test login to dashboard journey
   - Test medicine creation flow
   - Test distribution process

Test Coverage Target: Minimum 70%

---

8. DEPLOYMENT REQUIREMENTS

8.1 Backend Deployment

Deployment Options:

1. **Local/On-Premise Server:**
   - Package as JAR file
   - Run with Java 17+
   - Configure MySQL database
   - Set up reverse proxy (Nginx)
2. **Cloud Deployment (AWS/Azure/GCP):**
   - Deploy on Elastic Beanstalk/App Service/Cloud Run
   - Use managed MySQL (RDS/Azure Database/Cloud SQL)
   - Configure environment variables
   - Set up auto-scaling
3. **Containerization (Docker):**

dockerfile

```
FROM openjdk:17-slim
WORKDIR /app
COPY target/meditrack-0.0.1-SNAPSHOT.jar app.jar
EXPOSE 8080
ENTRYPOINT ["java", "-jar", "app.jar"]
```

8.2 Frontend Deployment

Deployment Options:

1. **Vercel/Netlify:**
   - Connect GitHub repository
   - Auto-deploy on push
   - Configure environment variables
   - Set up custom domain
2. **Nginx Static Hosting:**

```
npm run build
Copy dist/ contents to /var/www/html/
Configure Nginx to serve React app
```

3. **Docker Container:**

dockerfile

```
FROM node:18-alpine AS build
WORKDIR /app
COPY package*.json ./
RUN npm install
COPY . .
RUN npm run build

FROM nginx:alpine
COPY --from=build /app/dist /usr/share/nginx/html
EXPOSE 80
```

---

9. DOCUMENTATION REQUIREMENTS

9.1 API Documentation

- Use Swagger/OpenAPI for interactive API docs

- Document all endpoints with:
  - Request/response schemas
  - Authentication requirements
  - Example requests
  - Error responses

9.2 User Documentation

- User manual for each role (Admin, Supplier, Pharmacy)
- Quick start guide
- Video tutorials
- FAQ section
- Troubleshooting guide

9.3 Developer Documentation

- System architecture diagram
- Database schema documentation
- Setup instructions
- Contribution guidelines
- Code style guide

---

10. PERFORMANCE & OPTIMIZATION

10.1 Backend Optimization

- Implement database indexing on frequently queried fields
- Use pagination for large datasets
- Implement caching with Spring Cache (Redis optional)
- Optimize N+1 query problems with JOIN FETCH
- Use async processing for notifications and reports

10.2 Frontend Optimization

- Implement lazy loading for routes
- Use React.memo for expensive components
- Implement virtual scrolling for large lists
- Optimize images (WebP format, lazy loading)
- Use code splitting
- Implement service worker for offline capability

This comprehensive requirements document provides a complete blueprint for developing MediTrack with all necessary pages, APIs, database schema, and implementation guidance.