# Warehouse Management System - Complete Project Documentation

**Version:** 2.0
**Last Updated:** January 27, 2026
**Author:** WMS Development Team
**Technology Stack:** MERN (MongoDB, Express.js, React, Node.js)

---

## Table of Contents



---

# 1. Project Overview

## 1.1 Introduction

The Warehouse Management System (WMS) is a comprehensive full-stack web application designed to manage grain storage operations for warehouse owners. The system facilitates customer onboarding, storage allocation, quality inspection, payment processing, worker management, and real-time monitoring of warehouse operations.

## 1.2 Key Objectives

- **Efficient Storage Management:** Track grain storage across multiple warehouse locations
- **Customer Portal:** Enable customers to monitor their stored grains, payments, and transactions
- **Worker Operations:** Streamline daily tasks including vehicle weighing, quality checks, and loading operations
- **Financial Tracking:** Automated billing, payment collection, and loan management
- **Data Analytics:** Advanced visualization and ML-powered predictions for business insights
- **Multi-Language Support:** Seamless experience in multiple languages (English, Telugu)
- **AI Automation:** Intelligent alerts and reminders for timely transactions

## 1.3 Project Scope

- **Users Supported:** Warehouse Owners, Customers, Workers
- **Grain Types:** Rice, Wheat, Maize, Bajra, Other grains
- **Payment Methods:** Cash, UPI, Bank Transfer, Razorpay integration
- **Languages:** English, Telugu (with i18n framework)
- **Platforms:** Web-based (responsive design for mobile/tablet/desktop)

---

# 2. Technology Stack

## 2.1 Frontend Technologies

| Technology | Version | Purpose |
|---|---|---|
| React | 19.0.0 | UI framework with component-based architecture |
| Material-UI (MUI) | 7.2.1 | Modern UI component library |
| Recharts | 3.2.0 | Data visualization and charts |
| Axios | 1.8.1 | HTTP client for API requests |
| React Router DOM | 7.1.2 | Client-side routing |
| Socket.IO Client | 4.8.1 | Real-time bidirectional communication |
| i18next | Latest | Internationalization framework |
| react-i18next | Latest | React bindings for i18next |

## 2.2 Backend Technologies

| Technology | Version | Purpose |
|---|---|---|
| Node.js | 18.x+ | JavaScript runtime environment |
| Express.js | 4.18.2 | Web application framework |
| MongoDB | Latest | NoSQL database |
| Mongoose | 8.9.4 | MongoDB ODM |
| JWT | 9.0.2 | Authentication tokens |

| Technology | Version | Purpose |
| --- | --- | --- |
| bcryptjs | 2.4.3 | Password hashing |
| Razorpay | 2.9.4 | Payment gateway integration |
| Multer | 1.4.5 | File upload handling |
| ExcelJS | 4.4.0 | Excel file generation |
| Nodemailer | 6.9.16 | Email service |
| node-cron | 3.0.3 | Task scheduling |
| OpenAI SDK | Latest | AI-powered features |
| Twilio | 5.4.0 | SMS notifications |

## 2.3 Python Analytics Stack

| Technology | Purpose |
| --- | --- |
| Pandas | Data manipulation |
| NumPy | Numerical computing |
| Scikit-learn | Machine learning models |
| Matplotlib | Data visualization |
| Seaborn | Statistical visualizations |
| Dash/Streamlit | Interactive dashboards |

## 2.4 Development Tools

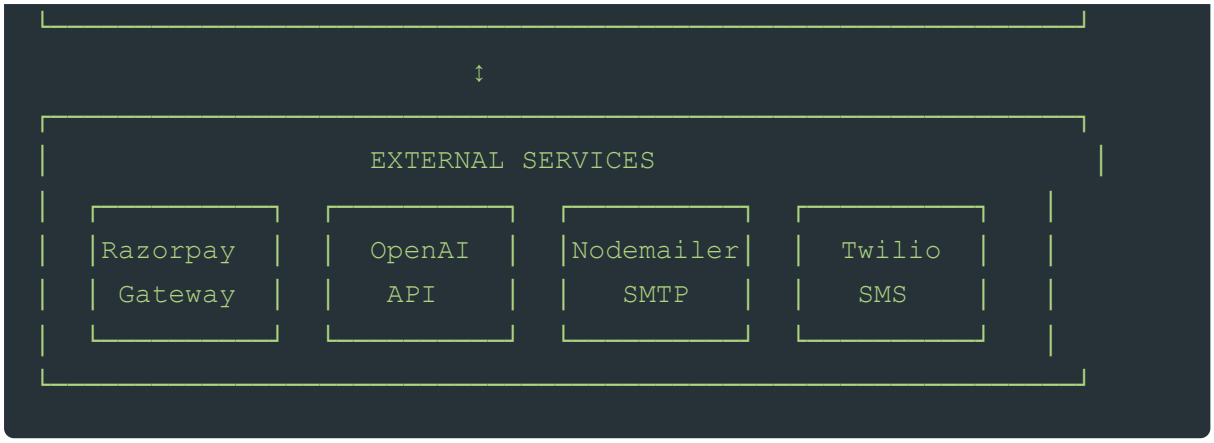
- Git - Version control
- VS Code - Code editor

- Postman - API testing
- MongoDB Compass - Database management
- Jupyter Notebook - ML model development

---

# 3. System Architecture

## 3.1 High-Level Architecture




## 3.2 Component Architecture

- **Frontend:** Component-based React architecture with hooks and context API
- **Backend:** RESTful API with Express middleware pipeline
- **Database:** MongoDB collections with Mongoose schemas
- **Real-time:** Socket.IO for live updates
- **ML Pipeline:** Python services for analytics and predictions
- **AI Integration:** OpenAI/Claude API for intelligent automation

---

# 4. User Roles & Permissions

## 4.1 Owner Role

**Description:** Warehouse owner with full administrative access

**Permissions:** - ✅ View comprehensive analytics dashboard - ✅ Manage all customers and workers - ✅ Configure warehouse layouts - ✅ Approve/reject customer applications - ✅ Set pricing and rental rates - ✅ Generate financial reports - ✅ Configure AI alert settings - ✅ Access ML predictions and insights - ✅ Change system language preferences - ✅ Full CRUD operations on all entities

**Dashboard Features:** - Real-time storage occupancy - Revenue analytics - Customer insights - Worker performance tracking - Payment collection status - Grain inventory visualization - ML-powered predictions - Multi-language interface

## 4.2 Customer Role

**Description:** Grain storage customer

**Permissions:** - ✅ View personal grain storage details - ✅ Track storage duration and costs - ✅ Make online payments - ✅ View transaction history - ✅ Download invoices and receipts - ✅ Request quality inspections - ✅ Communicate with warehouse staff - ✅ Access multi-language interface - ❌ Cannot view other customers' data - ❌ Cannot modify system settings

**Dashboard Features:** - My grains overview - Payment timeline - Storage cost calculator - Transaction history - Notifications panel - Quality inspection reports - Language preference settings

## 4.3 Worker Role

**Description:** Warehouse staff handling daily operations

**Permissions:** - ✅ Perform weighbridge operations - ✅ Conduct quality inspections - ✅ Manage vehicle queue - ✅ Update loading/unloading status - ✅ Mark daily tasks complete - ✅ View assigned work orders - ✅ Use language preference - ❌ Cannot access financial data - ❌ Cannot manage customers

**Dashboard Features:** - Daily task board - Weighbridge interface - Quality inspection forms - Vehicle queue manager - Loading operations tracker - Attendance tracker - Language selection

---

# 5. Core Features

## 5.1 Authentication & Authorization

**Login System:** - JWT-based authentication - Role-based access control (RBAC) - Secure password hashing with bcrypt - Session management - **NEW: Language Preference Selection at Login** - Remember me functionality

**Registration:** - Customer self-registration - Email verification - Mobile OTP verification (via Twilio) - Document upload during registration

**Security Features:** - HTTP-only cookies - CORS protection - Rate limiting - Input validation and sanitization - SQL injection prevention - XSS protection

## 5.2 Vehicle Management

**Features:** - Vehicle registration with image upload - Driver details management - Vehicle type categorization (Truck, Tempo, Tractor) - Load capacity tracking - Entry/exit timestamps - Queue management system

**Weighbridge Operations:** - Empty weight recording - Loaded weight recording - Net weight calculation - QR code generation for tracking - Weighbridge slip printing - Real-time weight updates

## 5.3 Warehouse Storage Management

**Storage Allocation:** - Dynamic warehouse layout configuration - Grid-based storage visualization - Storage location assignment - Capacity tracking per location - Grain type segregation - Storage duration monitoring

**Grain Inventory:** - Real-time stock levels - Grain type classification - Bag count tracking - Weight tracking (kg/tons) - Storage start/end dates - Automatic expiry alerts

## 5.4 Loan Management

**Features:** - Loan application processing - Interest rate calculation - EMI scheduling - Loan approval workflow - Repayment tracking - Loan portfolio overview - Default risk alerts

**Loan Types:** - Short-term loans (< 6 months) - Medium-term loans (6-12 months) - Long-term loans (> 12 months) - Interest rates: 8-15% per annum

## 5.5 Payment Processing

**Payment Methods:** - Cash payments - UPI/QR code - Bank transfer - Razorpay online payments - Cheque payments

**Features:** - Payment recording and tracking - Receipt generation (PDF) - Payment history - Due date reminders - Late payment penalties - Razorpay integration for

online payments - Payment analytics

## 5.6 Quality Inspection

**Inspection Parameters:** - Moisture content - Foreign material percentage - Damaged grain percentage - Color and appearance - Odor assessment - Pest infestation check

**Workflow:** 1. Customer requests inspection 2. Worker performs inspection 3. Results recorded with timestamp 4. Quality certificate generated 5. Customer notified 6. Report stored in document vault

## 5.7 Customer Portal Features

**My Grains Overview:** - Current storage details - Storage duration - Total cost calculation - Payment status - Quality reports

**Payment Module:** - View outstanding dues - Payment timeline - Make online payments - Download receipts - Payment history

**Notifications:** - Payment reminders - Quality inspection alerts - Storage expiry warnings - Promotional offers - System announcements

**Document Vault:** - Upload/download contracts - Storage receipts - Quality certificates - Payment invoices - Vehicle entry slips

## 5.8 Worker Dashboard Features

**Daily Task Board:** - Assigned tasks list - Task completion marking - Priority indicators - Deadline tracking - Performance metrics

**Weighbridge Operations:** - Vehicle weighing interface - Weight entry forms - Slip generation - Historical weighing data

**Quality Inspection:** - Inspection form interface - Parameter entry - Photo upload - Report generation

**Loading Operations:** - Loading schedule - Vehicle assignment - Progress tracking - Completion confirmation

**Vehicle Queue Manager:** - Real-time queue visualization - Priority management - Estimated wait time - Vehicle check-in/check-out

### 5.9 Real-time Features

**WebSocket Implementation (Socket.IO):** - Live storage updates - Real-time notifications - Vehicle queue updates - Payment confirmations - Worker task assignments - Alert broadcasts

**Push Notifications:** - Browser notifications - In-app notification center - Email notifications - SMS alerts (via Twilio)

### 5.10 Alert Center

**Alert Types:** - Payment due reminders - Storage expiry warnings - Quality inspection schedules - Loan repayment reminders - System maintenance notifications - Critical alerts (pest detection, moisture issues)

**Alert Configuration:** - Custom alert rules - Recipient selection - Alert frequency settings - Priority levels - Escalation rules

### 5.11 Inventory Spot Check

**Features:** - Random inventory verification - Discrepancy reporting - Stock adjustment - Audit trail - Reconciliation reports

---

## 6. Analytics & Visualization Module

### 6.1 Overview

The enhanced analytics module provides warehouse owners with comprehensive visual insights and machine learning-powered predictions to make data-driven decisions.

### 6.2 Interactive Visualizations

### 6.2.1 Grain-Based Analytics

**Visualization Components:** - **Grain Type Distribution** - Pie chart showing percentage of each grain type - **Grain Quantity Trends** - Line chart tracking storage volumes over time - **Grain Revenue Analysis** - Bar chart comparing revenue by grain type - **Grain Turnover Rate** - Heat map showing grain movement frequency

**Filters Available:** - Date range selection - Grain type filter - Storage location filter - Customer segment filter

**Key Metrics:** - Total bags stored per grain type - Total weight (kg/tons) per grain type - Average storage duration per grain type - Revenue generated per grain type - Current inventory levels

### 6.2.2 Storage Duration Analytics

**Visualization Components:** - **Duration Distribution** - Histogram showing storage period patterns - **Duration Category Chart** - Donut chart (Short/Medium/Long-term storage) - **Duration vs Revenue** - Scatter plot correlation analysis - **Storage Timeline** - Gantt chart showing active storage periods

**Insights Provided:** - Average storage duration across all customers - Seasonal storage patterns - Duration-based pricing optimization - Customer retention by storage length

### 6.2.3 Customer Analytics

**Visualization Components:** - **Customer In/Out Flow** - Line chart tracking new vs exiting customers - **Customer Segmentation** - Bubble chart by storage value and frequency - **Customer Lifetime Value** - Bar chart ranking top customers - **Geographic Distribution** - Map visualization of customer locations

**Metrics Tracked:** - New customer acquisition rate - Customer churn rate - Active customers vs dormant - Average customer lifetime value - Repeat customer percentage

### 6.2.4 Warehouse Capacity Visualization

**Dashboard Elements:** - **Real-time Occupancy Gauge** - Circular gauge showing current capacity utilization - **Storage Location Heatmap** - Grid view with color-coded

occupancy - **Capacity Trends** - Area chart showing capacity over time - **Utilization by Section** - Stacked bar chart

**Features:** - Interactive warehouse layout map - Click-to-drill-down to specific locations - Capacity alerts and warnings - Predictive capacity planning

## 6.3 Machine Learning Integration

### 6.3.1 Model Implementation

**Approach:** Logistic Regression Classifier

**Rationale:** - Warehouse operations typically have smaller customer datasets - Logistic regression performs well with limited data - Fast training and inference times - Interpretable results for business decisions - Low computational requirements

**Model Architecture:**

```
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler

# Model configuration
model = LogisticRegression(
    C=1.0,
    max_iter=1000,
    solver='lbfgs',
    random_state=42
)

# Feature scaling
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X_train)
model.fit(X_scaled, y_train)
```

### 6.3.2 Prediction Capabilities

**1. Customer Churn Prediction** - **Input Features:** - Storage duration history - Payment timeliness - Storage value trends - Interaction frequency - Quality inspection frequency

- **Output:** Probability of customer leaving (0-1 scale)
- **Business Action:** Proactive retention campaigns for high-risk customers

**2. Payment Default Risk** - **Input Features:** - Payment history - Outstanding balance - Days past due - Loan status - Storage value

- **Output:** Risk category (Low/Medium/High)
- **Business Action:** Early intervention for high-risk accounts

**3. Storage Duration Prediction** - **Input Features:** - Grain type - Customer history - Seasonal patterns - Storage quantity - Market prices

- **Output:** Expected storage duration category
- **Business Action:** Capacity planning and pricing optimization

**4. Revenue Forecasting** - **Input Features:** - Historical revenue data - Current storage levels - Seasonal trends - Market conditions - Customer growth rate

- **Output:** Expected monthly revenue range
- **Business Action:** Budget planning and resource allocation

### 6.3.3 Model Performance Metrics

All ML models are evaluated using classification metrics: - **Accuracy:** Overall prediction correctness - **Precision:** Percentage of correct positive predictions - **Recall:** Percentage of actual positives captured - **F1-Score:** Balanced precision-recall metric - **Confusion Matrix:** Visual breakdown of prediction errors - **Support:** Sample count per class

### 6.3.4 Model Training Pipeline

```
1. Data Collection
   └─> Extract from CUSTOMER_ACTIVITIES.csv & GRAIN_MOVEMENTS.csv

2. Data Preprocessing
   ├─> Handle missing values
   ├─> Encode categorical variables
   ├─> Feature scaling (StandardScaler)
   └─> Train-test split (80-20)
```

```
3. Model Training
   ├─> Logistic Regression with cross-validation
   ├─> Hyperparameter tuning
   └─> Performance evaluation

4. Model Deployment
   ├─> Save model as .pkl file
   ├─> Load in Node.js backend via Python script
   └─> Real-time predictions via API

5. Monitoring
   ├─> Track prediction accuracy
   ├─> Retrain monthly with new data
   └─> A/B testing for model improvements
```

## 6.4 Dashboard Implementation

### Owner Analytics Dashboard Components:

```
// Analytics Dashboard Structure
<OwnerAnalyticsDashboard>
  <MetricsOverview>
    - Total Revenue
    - Active Customers
    - Storage Utilization
    - Payment Collection Rate
  </MetricsOverview>

  <VisualizationGrid>
    <GrainDistributionChart type="pie" />
    <StorageDurationChart type="bar" />
    <CustomerFlowChart type="line" />
    <CapacityHeatmap type="heatmap" />
  </VisualizationGrid>

  <MLPredictions>
    <ChurnRiskAlerts />
    <PaymentDefaultWarnings />
    <RevenueForecast />
  </MLPredictions>
```

```
  <InteractiveFilters>
    <DateRangePicker />
    <GrainTypeSelector />
    <CustomerSegmentFilter />
  </InteractiveFilters>
</OwnerAnalyticsDashboard>
```

**Technology Stack for Visualizations:** - **Recharts** - Primary charting library - **D3.js** - Custom visualizations - **Chart.js** - Alternative charts - **React Grid Layout** - Draggable/resizable dashboard

**Data Refresh:** - Real-time updates via WebSocket - Auto-refresh every 30 seconds - Manual refresh button - Last updated timestamp

## 6.5 Analytics API Endpoints

### GET /api/analytics/grain-distribution

```
Response:
{
  "grains": [
    {"type": "Rice", "bags": 1500, "weight_kg": 75000, "percentage": 4
    {"type": "Wheat", "bags": 1200, "weight_kg": 60000, "percentage":
    {"type": "Maize", "bags": 800, "weight_kg": 40000, "percentage": 2
    {"type": "Others", "bags": 250, "weight_kg": 12500, "percentage":
  ],
  "total_bags": 3750,
  "total_weight_kg": 187500
}
```

### GET /api/analytics/storage-duration

```
Response:
{
  "categories": {
    "short_term": 45,     // < 30 days
    "medium_term": 120,   // 30-90 days
    "long_term": 85       // > 90 days
  },
  "average_duration_days": 62,
```

```
  "duration_trends": [
    {"month": "Jan", "avg_days": 58},
    {"month": "Feb", "avg_days": 65}
  ]
}
```

### GET /api/analytics/customer-flow

```
Response:
{
  "monthly_data": [
    {
      "month": "2026-01",
      "customers_in": 25,
      "customers_out": 18,
      "net_change": 7,
      "active_total": 142
    }
  ],
  "churn_rate": 12.7,
  "retention_rate": 87.3
}
```

### POST /api/analytics/ml-predict

```
Request:
{
  "model": "churn_prediction",
  "customer_id": "CUST001",
  "features": {
    "storage_duration_avg": 45,
    "payment_timeliness": 0.85,
    "interaction_frequency": 12
  }
}

Response:
{
  "prediction": "low_risk",
  "probability": 0.23,
  "confidence": 0.87,
```

```
  "recommendation": "No immediate action required"
}
```

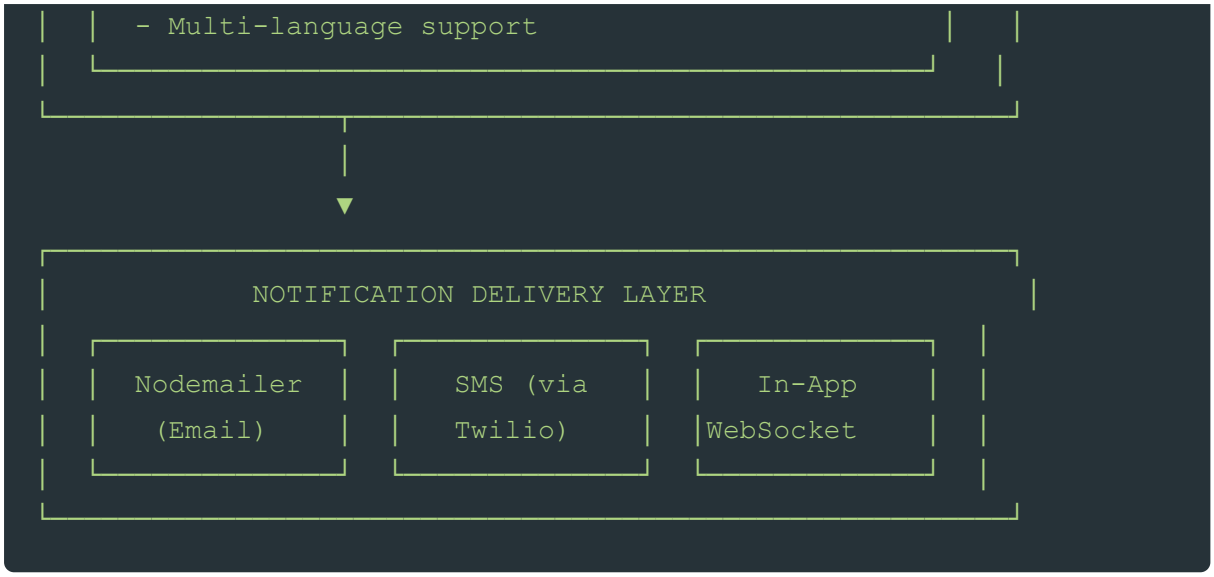
---

# 7. AI-Powered Alert System

## 7.1 Overview

An intelligent automation system that monitors customer activities and sends timely reminders for pending transactions, payments, and important deadlines using AI-powered natural language generation.

## 7.2 System Architecture




## 7.3 Implementation Details

### 7.3.1 Scheduler Configuration (node-cron)

#### Installation:

```
npm install node-cron
```

#### Cron Job Setup:

```
// server/jobs/alertScheduler.js
const cron = require('node-cron');
const { runAIAlertJob } = require('./aiAlertJob');

// Run every hour at the top of the hour
cron.schedule('0 * * * *', async () => {
  console.log('🤖 Running AI Alert Job at', new Date());
  await runAIAlertJob();
});

// Run daily at 9 AM for morning reminders
cron.schedule('0 9 * * *', async () => {
  console.log('🌅 Morning reminder batch processing');
  await runAIAlertJob('morning');
});

// Run daily at 6 PM for evening reminders
```

```
cron.schedule('0 18 * * *', async () => {
  console.log('🌆 Evening reminder batch processing');
  await runAIAlertJob('evening');
});
```

**Cron Schedule Patterns:** - `0 * * * *` - Every hour - `0 9 * * *` - Daily at 9:00 AM - `0 18 * * *` - Daily at 6:00 PM - `0 0 * * 0` - Weekly on Sunday midnight - `0 0 1 * *` - Monthly on 1st day

### 7.3.2 AI Agent Implementation

#### Option 1: OpenAI Integration

```
// server/services/aiAlertService.js
const OpenAI = require('openai');

const openai = new OpenAI({
  apiKey: process.env.OPENAI_API_KEY
});

async function generateAlertMessage(alertData) {
  const { customerName, type, dueDate, amount, language } = alertData;

  const systemPrompt = `You are a helpful warehouse management assista
  Generate a polite, professional reminder message in ${language} lang

  const userPrompt = `Generate a reminder for:
  - Customer: ${customerName}
  - Alert Type: ${type}
  - Due Date: ${dueDate}
  - Amount: ₹${amount}

  Keep it concise (2-3 sentences), friendly, and actionable.`;

  const completion = await openai.chat.completions.create({
    model: "gpt-4",
    messages: [
      { role: "system", content: systemPrompt },
      { role: "user", content: userPrompt }
    ],
    temperature: 0.7,
```

```
    max_tokens: 150
  });

  return completion.choices[0].message.content;
}
```

### Option 2: Claude (Anthropic) Integration

```
const Anthropic = require('@anthropic-ai/sdk');

const claude = new Anthropic({
  apiKey: process.env.CLAUDE_API_KEY
});

async function generateAlertMessage(alertData) {
  const { customerName, type, dueDate, amount, language } = alertData;

  const message = await claude.messages.create({
    model: "claude-3-sonnet-20240229",
    max_tokens: 150,
    messages: [{
      role: "user",
      content: `Generate a polite payment reminder in ${language} for
      ${customerName}. Payment of ₹${amount} is due on ${dueDate}.
      Type: ${type}. Keep it professional and concise.`
    }]
  });

  return message.content[0].text;
}
```

## 7.3.3 Alert Detection Logic

```
// server/jobs/aiAlertJob.js
const User = require('../models/User');
const Transaction = require('../models/Transaction');
const Loan = require('../models/Loan');
const { generateAlertMessage } = require('../services/aiAlertService')
const { sendEmail } = require('../utils/emailService');
const { sendSMS } = require('../utils/smsService');
```

```
async function runAIAlertJob(timeSlot = 'default') {
  try {
    // 1. Find customers with pending payments
    const pendingPayments = await Transaction.find({
      payment_status: 'pending',
      due_date: {
        $gte: new Date(),
        $lte: new Date(Date.now() + 7 * 24 * 60 * 60 * 1000) // Next 7
      }
    }).populate('customer_id');

    // 2. Find customers with upcoming loan repayments
    const upcomingLoanPayments = await Loan.find({
      status: 'active',
      next_payment_date: {
        $gte: new Date(),
        $lte: new Date(Date.now() + 3 * 24 * 60 * 60 * 1000) // Next 3
      }
    }).populate('customer_id');

    // 3. Process each alert
    for (const transaction of pendingPayments) {
      const customer = transaction.customer_id;

      // Generate AI-powered message
      const message = await generateAlertMessage({
        customerName: customer.name,
        type: 'payment_reminder',
        dueDate: transaction.due_date,
        amount: transaction.total_amount,
        language: customer.preferred_language || 'english'
      });

      // Send via email
      await sendEmail({
        to: customer.email,
        subject: 'Payment Reminder - WMS',
        text: message,
        html: `<div style="padding: 20px;">${message}</div>`
      });

      // Send via SMS (optional)
      if (customer.mobile) {
```

```
        await sendSMS({
          to: customer.mobile,
          message: message
        });
      }

      // Log alert sent
      console.log(`✅ Alert sent to ${customer.name} (${customer.email
    }

    // Process loan reminders similarly
    for (const loan of upcomingLoanPayments) {
      // ... similar process
    }

    console.log(`🎯 AI Alert Job completed. Sent ${pendingPayments.len

  } catch (error) {
    console.error('❌ AI Alert Job failed:', error);
  }
}

module.exports = { runAIAlertJob };
```

### 7.3.4 Email Service (Nodemailer)

#### Configuration:

```
// server/utils/emailService.js
const nodemailer = require('nodemailer');

const transporter = nodemailer.createTransport({
  host: process.env.SMTP_HOST,       // e.g., smtp.gmail.com
  port: process.env.SMTP_PORT,       // 587 or 465
  secure: process.env.SMTP_SECURE === 'true', // true for 465, false f
  auth: {
    user: process.env.SMTP_USER,     // your email
    pass: process.env.SMTP_PASS      // app password
  }
});

async function sendEmail({ to, subject, text, html }) {
```

```
  try {
    const info = await transporter.sendMail({
      from: `"WMS Alerts" <${process.env.SMTP_USER}>`,
      to,
      subject,
      text,
      html
    });

    console.log('📧 Email sent:', info.messageId);
    return { success: true, messageId: info.messageId };
  } catch (error) {
    console.error('❌ Email failed:', error);
    return { success: false, error: error.message };
  }
}

module.exports = { sendEmail };
```

## Email Templates:

```
// Payment Reminder Email Template
function getPaymentReminderTemplate(data) {
  return `
    <!DOCTYPE html>
    <html>
    <head>
      <style>
        body { font-family: Arial, sans-serif; }
        .container { max-width: 600px; margin: 0 auto; padding: 20px;
        .header { background: #2196F3; color: white; padding: 20px; te
        .content { padding: 20px; background: #f9f9f9; }
        .amount { font-size: 24px; font-weight: bold; color: #f44336;
        .button { background: #4CAF50; color: white; padding: 12px 30p
                  text-decoration: none; border-radius: 5px; display:
      </style>
    </head>
    <body>
      <div class="container">
        <div class="header">
          <h1>Payment Reminder</h1>
        </div>
```

```
        <div class="content">
          <p>Dear ${data.customerName},</p>
          <p>${data.aiGeneratedMessage}</p>
          <p class="amount">Amount Due: ₹${data.amount}</p>
          <p>Due Date: ${data.dueDate}</p>
          <br>
          <a href="${data.paymentLink}" class="button">Pay Now</a>
        </div>
      </div>
    </body>
    </html>
  `;
}
```

### 7.3.5 SMS Integration (Twilio)

```
// server/utils/smsService.js
const twilio = require('twilio');

const client = twilio(
  process.env.TWILIO_ACCOUNT_SID,
  process.env.TWILIO_AUTH_TOKEN
);

async function sendSMS({ to, message }) {
  try {
    const sms = await client.messages.create({
      body: message,
      from: process.env.TWILIO_PHONE_NUMBER,
      to: to
    });

    console.log('📱 SMS sent:', sms.sid);
    return { success: true, sid: sms.sid };
  } catch (error) {
    console.error('❌ SMS failed:', error);
    return { success: false, error: error.message };
  }
}

module.exports = { sendSMS };
```

## 7.4 Alert Types & Rules

**1. Payment Due Reminders** - **Trigger:** 7 days before due date, 3 days before, 1 day before, on due date - **Priority:** High - **Channels:** Email + SMS + In-App - **AI Context:** Include payment amount, due date, payment methods

**2. Storage Expiry Warnings** - **Trigger:** 14 days before contract end - **Priority:** Medium - **Channels:** Email + In-App - **AI Context:** Storage details, renewal options

**3. Loan Repayment Reminders** - **Trigger:** 5 days before EMI due date - **Priority:** High - **Channels:** Email + SMS - **AI Context:** EMI amount, outstanding balance

**4. Quality Inspection Notifications** - **Trigger:** 30 days after last inspection - **Priority:** Medium - **Channels:** Email + In-App - **AI Context:** Last inspection date, recommended frequency

**5. Document Expiry Alerts** - **Trigger:** 10 days before contract/insurance expiry - **Priority:** Medium - **Channels:** Email - **AI Context:** Document type, expiry date, renewal process

## 7.5 Configuration & Management

**Owner Alert Center Dashboard:**

```
<AlertCenter>
  <AlertRulesConfig>
    - Enable/Disable alert types
    - Set trigger thresholds (days before)
    - Configure notification channels
    - Set business hours for sending
  </AlertRulesConfig>

  <AlertHistory>
    - View sent alerts
    - Delivery status tracking
    - Customer response tracking
    - Alert effectiveness analytics
  </AlertHistory>

  <AISettings>
    - Select AI provider (OpenAI/Claude)
```

```
    - Configure message tone (formal/friendly)
    - Language preferences
    - Template customization
  </AISettings>
</AlertCenter>
```

**Environment Variables:**

```
# AI Configuration
OPENAI_API_KEY=sk-...
CLAUDE_API_KEY=sk-ant-...
AI_PROVIDER=openai  # or 'claude'

# Email Configuration
SMTP_HOST=smtp.gmail.com
SMTP_PORT=587
SMTP_SECURE=false
SMTP_USER=your-email@gmail.com
SMTP_PASS=your-app-password

# SMS Configuration
TWILIO_ACCOUNT_SID=AC...
TWILIO_AUTH_TOKEN=...
TWILIO_PHONE_NUMBER=+1234567890

# Cron Settings
ENABLE_ALERT_SCHEDULER=true
ALERT_TIMEZONE=Asia/Kolkata
```

## 7.6 Performance & Monitoring

**Metrics Tracked:** - Total alerts sent per day - Delivery success rate (email/SMS) - Customer response rate - AI generation time - Cost per alert (API usage)

**Error Handling:** - Retry failed email/SMS attempts (3 retries with exponential backoff) - Fallback to template messages if AI fails - Log all errors to monitoring system - Alert admin for critical failures

**Rate Limiting:** - Max 100 emails per hour (SMTP limits) - Max 50 SMS per hour (cost optimization) - Batch processing for large customer base

# 8. Multi-Language Support (i18n)

## 8.1 Overview

Implement comprehensive internationalization (i18n) to provide a seamless user experience in multiple languages. Users can select their preferred language at login, and the entire application interface dynamically adapts to display all text in the chosen language.

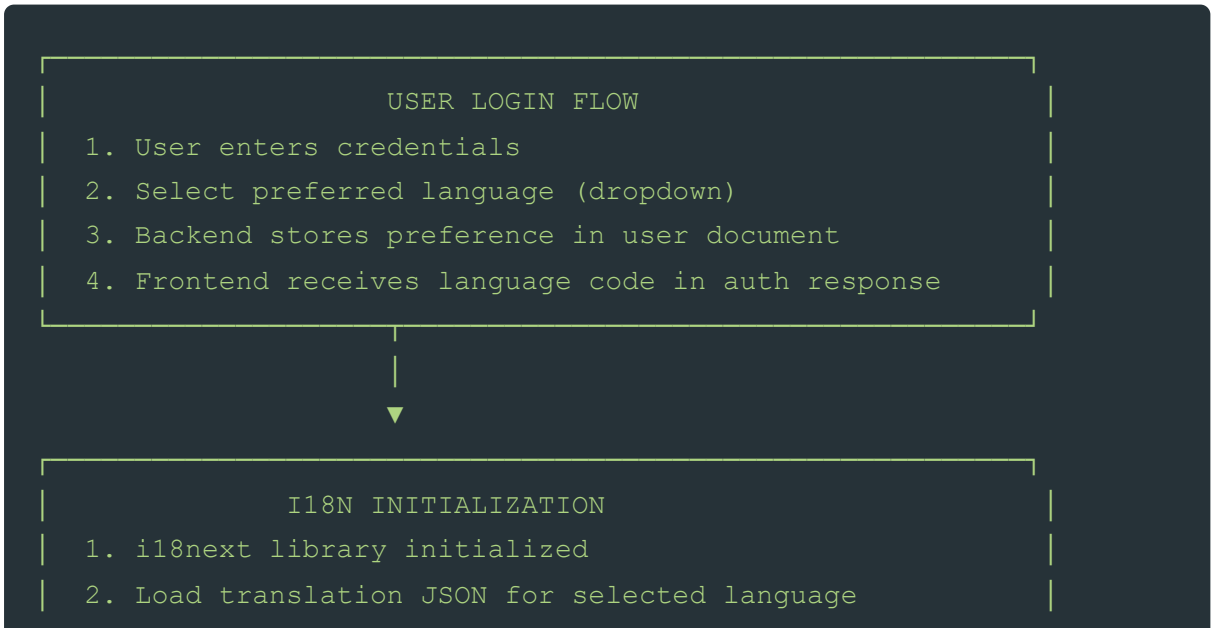
## 8.2 Supported Languages

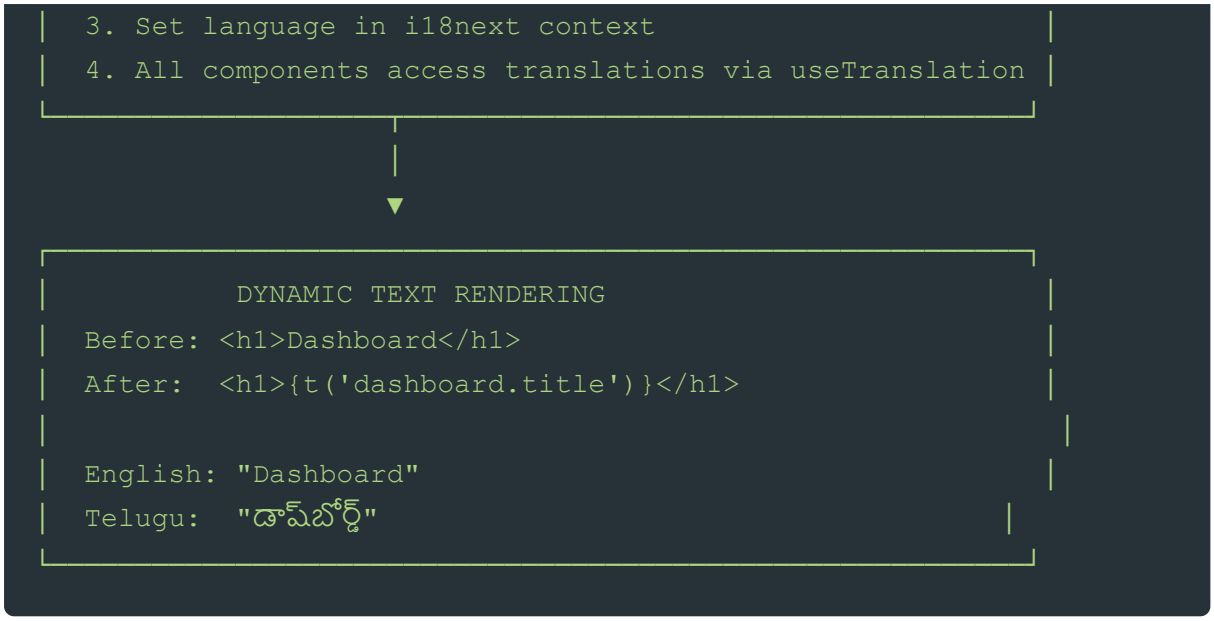
- **English (en)** - Default language
- **Telugu (te)** - Regional language support
- **Extensible** - Framework ready for adding more languages (Hindi, Tamil, etc.)

## 8.3 Core Concept

**One-Sentence Summary:**
Store the user's preferred language at login, load the language dictionary dynamically, and render all UI text via translation keys instead of hard-coded strings.

## 8.4 Implementation Architecture






## 8.5 Technology Stack

### NPM Packages:

```
{
  "dependencies": {
    "i18next": "^23.7.0",
    "react-i18next": "^14.0.0",
    "i18next-http-backend": "^2.4.0",
    "i18next-browser-languagedetector": "^7.2.0"
  }
}
```

### Installation:

```
npm install i18next react-i18next i18next-http-backend i18next-browser
```

## 8.6 Backend Implementation

### 8.6.1 User Schema Update

```
// server/models/User.js
const userSchema = new mongoose.Schema({
  name: String,
```

```
  email: String,
  password: String,
  role: String,
  preferred_language: {
    type: String,
    enum: ['en', 'te'],
    default: 'en'
  },
  // ... other fields
});
```

### 8.6.2 Login API Modification

```
// server/routes/auth.js
router.post('/login', async (req, res) => {
  try {
    const { email, password, language } = req.body;

    const user = await User.findOne({ email });
    if (!user) {
      return res.status(401).json({ message: 'Invalid credentials' });
    }

    const isMatch = await bcrypt.compare(password, user.password);
    if (!isMatch) {
      return res.status(401).json({ message: 'Invalid credentials' });
    }

    // Update preferred language
    if (language) {
      user.preferred_language = language;
      await user.save();
    }

    const token = jwt.sign(
      { userId: user._id, role: user.role },
      process.env.JWT_SECRET,
      { expiresIn: '7d' }
    );

    res.json({
      token,
```

```
      user: {
        id: user._id,
        name: user.name,
        email: user.email,
        role: user.role,
        preferred_language: user.preferred_language  // Return languag
      }
    });
  } catch (error) {
    res.status(500).json({ error: error.message });
  }
});
```

## 8.7 Frontend Implementation

### 8.7.1 i18n Configuration

```
// client/src/i18n.js
import i18n from 'i18next';
import { initReactI18next } from 'react-i18next';
import HttpBackend from 'i18next-http-backend';
import LanguageDetector from 'i18next-browser-languagedetector';

i18n
  .use(HttpBackend)  // Load translations via HTTP
  .use(LanguageDetector)  // Detect user language
  .use(initReactI18next)  // Pass i18n to React
  .init({
    fallbackLng: 'en',
    debug: false,

    interpolation: {
      escapeValue: false  // React already escapes
    },

    backend: {
      loadPath: '/locales/{{lng}}/{{ns}}.json'  // Translation file pa
    },

    ns: ['common', 'dashboard', 'auth'],  // Namespaces
    defaultNS: 'common'
```

```
  });

export default i18n;
```

### 8.7.2 App Initialization

```
// client/src/index.js
import React from 'react';
import ReactDOM from 'react-dom/client';
import App from './App';
import './i18n';  // Import i18n config

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
  <React.StrictMode>
    <App />
  </React.StrictMode>
);
```

### 8.7.3 Login Component

```
// client/src/pages/Login.js
import React, { useState } from 'react';
import { useTranslation } from 'react-i18next';
import axios from 'axios';

function Login() {
  const { t, i18n } = useTranslation();
  const [formData, setFormData] = useState({
    email: '',
    password: '',
    language: 'en'
  });

  const handleLogin = async (e) => {
    e.preventDefault();

    try {
      const response = await axios.post('/api/auth/login', formData);
```

```
      // Set language in i18n
      i18n.changeLanguage(response.data.user.preferred_language);

      // Store auth data
      localStorage.setItem('token', response.data.token);
      localStorage.setItem('language', response.data.user.preferred_la

      // Redirect to dashboard
      window.location.href = '/dashboard';
    } catch (error) {
      alert(t('auth.loginFailed'));
    }
  };

  return (
    <div className="login-container">
      <h1>{t('auth.loginTitle')}</h1>
      <form onSubmit={handleLogin}>
        <input
          type="email"
          placeholder={t('auth.emailPlaceholder')}
          value={formData.email}
          onChange={(e) => setFormData({ ...formData, email: e.target.
        />

        <input
          type="password"
          placeholder={t('auth.passwordPlaceholder')}
          value={formData.password}
          onChange={(e) => setFormData({ ...formData, password: e.targ
        />

        <select
          value={formData.language}
          onChange={(e) => setFormData({ ...formData, language: e.targ
        >
          <option value="en">English</option>
          <option value="te">తెలుగు (Telugu)</option>
        </select>

        <button type="submit">{t('auth.loginButton')}</button>
      </form>
    </div>
```

```
  );
}

export default Login;
```

### 8.7.4 Dashboard Component Example

```
// client/src/pages/OwnerDashboard.js
import React from 'react';
import { useTranslation } from 'react-i18next';

function OwnerDashboard() {
  const { t } = useTranslation('dashboard');

  return (
    <div>
      <h1>{t('title')}</h1>

      <div className="metrics">
        <div className="metric-card">
          <h3>{t('metrics.totalRevenue')}</h3>
          <p>₹1,250,000</p>
        </div>

        <div className="metric-card">
          <h3>{t('metrics.activeCustomers')}</h3>
          <p>142</p>
        </div>

        <div className="metric-card">
          <h3>{t('metrics.storageUtilization')}</h3>
          <p>78%</p>
        </div>
      </div>

      <button>{t('actions.addCustomer')}</button>
      <button>{t('actions.generateReport')}</button>
    </div>
  );
}
```

```
export default OwnerDashboard;
```

## 8.8 Translation Files Structure

### Directory Structure:

```
client/
  public/
    locales/
      en/
        common.json
        dashboard.json
        auth.json
        customer.json
        worker.json
      te/
        common.json
        dashboard.json
        auth.json
        customer.json
        worker.json
```

### English Translation (en/common.json):

```
{
  "appName": "Warehouse Management System",
  "navigation": {
    "dashboard": "Dashboard",
    "customers": "Customers",
    "inventory": "Inventory",
    "payments": "Payments",
    "reports": "Reports",
    "settings": "Settings"
  },
  "common": {
    "save": "Save",
    "cancel": "Cancel",
    "delete": "Delete",
    "edit": "Edit",
```

```
    "search": "Search",
    "filter": "Filter",
    "export": "Export",
    "loading": "Loading...",
    "noData": "No data available"
  },
  "messages": {
    "saveSuccess": "Saved successfully",
    "deleteConfirm": "Are you sure you want to delete?",
    "error": "An error occurred"
  }
}
```

### Telugu Translation (te/common.json):

```
{
  "appName": "గిడ్డంగి నిర్వహణ వ్యవస్థ",
  "navigation": {
    "dashboard": "డాష్‌బోర్డ్",
    "customers": "కస్టమర్లు",
    "inventory": "ఇన్వెంటరీ",
    "payments": "చెల్లింపులు",
    "reports": "నివేదికలు",
    "settings": "సెట్టింగ్‌లు"
  },
  "common": {
    "save": "సేవ్ చేయండి",
    "cancel": "రద్దు చేయండి",
    "delete": "తొలగించండి",
    "edit": "సవరించండి",
    "search": "వెతకండి",
    "filter": "ఫిల్టర్",
    "export": "ఎగుమతి చేయండి",
    "loading": "లోడ్ అవుతోంది...",
    "noData": "డేటా అందుబాటులో లేదు"
  },
  "messages": {
    "saveSuccess": "విజయవంతంగా సేవ్ చేయబడింది",
    "deleteConfirm": "మీరు ఖచ్చితంగా తొలగించాలనుకుంటున్నారా?",
    "error": "లోపం సంభవించింది"
```

```
  }
}
```

### English Dashboard (en/dashboard.json):

```
{
  "title": "Dashboard",
  "welcome": "Welcome back",
  "metrics": {
    "totalRevenue": "Total Revenue",
    "activeCustomers": "Active Customers",
    "storageUtilization": "Storage Utilization",
    "pendingPayments": "Pending Payments"
  },
  "actions": {
    "addCustomer": "Add Customer",
    "generateReport": "Generate Report",
    "viewAnalytics": "View Analytics"
  },
  "charts": {
    "grainDistribution": "Grain Distribution",
    "revenueOverTime": "Revenue Over Time",
    "customerGrowth": "Customer Growth"
  }
}
```

### Telugu Dashboard (te/dashboard.json):

```
{
  "title": "డాష్‌బోర్డ్",
  "welcome": "తిరిగి స్వాగతం",
  "metrics": {
    "totalRevenue": "మొత్తం ఆదాయం",
    "activeCustomers": "క్రియాశీల కస్టమర్లు",
    "storageUtilization": "నిల్వ వినియోగం",
    "pendingPayments": "పెండింగ్ చెల్లింపులు"
  },
  "actions": {
    "addCustomer": "కస్టమర్‌ను జోడించండి",
    "generateReport": "నివేదిక రూపొందించండి",
    "viewAnalytics": "విశ్లేషణలు చూడండి"
```

```
  },
  "charts": {
    "grainDistribution": "ధాన్యం పంపిణీ",
    "revenueOverTime": "కాలక్రమేణా ఆదాయం",
    "customerGrowth": "కస్టమర్ వృద్ధి"
  }
}
```

## 8.9 Language Switcher Component

```
// client/src/components/LanguageSwitcher.js
import React from 'react';
import { useTranslation } from 'react-i18next';
import { Select, MenuItem } from '@mui/material';

function LanguageSwitcher() {
  const { i18n } = useTranslation();

  const changeLanguage = async (lng) => {
    // Update i18n
    i18n.changeLanguage(lng);

    // Save to localStorage
    localStorage.setItem('language', lng);

    // Update in backend
    try {
      await axios.put('/api/users/update-language', {
        language: lng
      }, {
        headers: {
          Authorization: `Bearer ${localStorage.getItem('token')}`
        }
      });
    } catch (error) {
      console.error('Failed to update language preference');
    }
  };

  return (
    <Select
```

```
      value={i18n.language}
      onChange={(e) => changeLanguage(e.target.value)}
      size="small"
    >
      <MenuItem value="en">English</MenuItem>
      <MenuItem value="te">తెలుగు</MenuItem>
    </Select>
  );
}

export default LanguageSwitcher;
```

## 8.10 Complete Translation Coverage

**Components to Translate:** 1. **Navigation Menu** - All menu items 2. **Dashboard** - Metrics, charts, buttons 3. **Forms** - Labels, placeholders, error messages 4. **Tables** - Column headers, actions 5. **Modals** - Titles, confirmation messages 6. **Notifications** - Success/error messages 7. **Buttons** - All action buttons 8. **Tooltips** - Help text 9. **Validation Messages** - Form validation errors 10. **Date/Time Formats** - Localized formatting

### Example Form Translation:

```
<TextField
  label={t('customer.form.name')}
  placeholder={t('customer.form.namePlaceholder')}
  error={!!errors.name}
  helperText={errors.name && t('customer.form.nameRequired')}
/>
```

## 8.11 Number & Date Formatting

### Using Intl API for Localization:

```
import { useTranslation } from 'react-i18next';

function PriceDisplay({ amount }) {
  const { i18n } = useTranslation();
```

```
  const formattedPrice = new Intl.NumberFormat(
    i18n.language === 'te' ? 'te-IN' : 'en-IN',
    { style: 'currency', currency: 'INR' }
  ).format(amount);

  return <span>{formattedPrice}</span>;
}

function DateDisplay({ date }) {
  const { i18n } = useTranslation();

  const formattedDate = new Intl.DateTimeFormat(
    i18n.language === 'te' ? 'te-IN' : 'en-IN',
    { year: 'numeric', month: 'long', day: 'numeric' }
  ).format(new Date(date));

  return <span>{formattedDate}</span>;
}
```

## 8.12 RTL Support (Future Enhancement)

For languages like Arabic/Urdu (if added later):

```
// client/src/App.js
import { useEffect } from 'react';
import { useTranslation } from 'react-i18next';

function App() {
  const { i18n } = useTranslation();

  useEffect(() => {
    const isRTL = ['ar', 'ur'].includes(i18n.language);
    document.dir = isRTL ? 'rtl' : 'ltr';
  }, [i18n.language]);

  return <div>...</div>;
}
```

## 8.13 Best Practices

### 1. Use Translation Keys:

```
// ❌ Bad
<h1>Dashboard</h1>

// ✅ Good
<h1>{t('dashboard.title')}</h1>
```

**2. Namespace Organization:** - `common.json` - Shared terms (save, cancel, delete) - `dashboard.json` - Dashboard-specific terms - `auth.json` - Authentication terms - `customer.json` - Customer module terms - `worker.json` - Worker module terms

### 3. Plural Forms:

```
{
  "customer_one": "{{count}} customer",
  "customer_other": "{{count}} customers"
}
```

```
t('customer', { count: 5 })  // "5 customers"
```

### 4. Interpolation:

```
{
  "welcome": "Welcome, {{name}}!"
}
```

```
t('welcome', { name: 'John' })  // "Welcome, John!"
```

## 8.14 Performance Optimization

### Lazy Loading Translations:

```
i18n.init({
  backend: {
    loadPath: '/locales/{{lng}}/{{ns}}.json',
    requestOptions: {
```

```
      cache: 'default'  // Enable caching
    }
  },
  react: {
    useSuspense: true  // Use Suspense for loading
  }
});
```

**Pre-load Critical Namespaces:**

```
i18n.init({
  preload: ['en', 'te'],  // Pre-load these languages
  ns: ['common'],  // Load 'common' namespace immediately
  defaultNS: 'common'
});
```

---

# 9. Machine Learning Models

## 9.1 Overview

Three classification models have been developed using Python and scikit-learn to provide predictive insights:

1. **Price Category Prediction** - Predicts grain sale price range (Low/Medium/High)
2. **Profit/Loss Classification** - Predicts whether customer will make profit or loss
3. **Storage Duration Prediction** - Predicts storage period category (Short/Medium/Long-term)

## 9.2 Model 1: Price Category Prediction

**File:** `wms-analytics/price_prediction.ipynb`

**Objective:** Predict grain sale price category based on storage patterns and grain characteristics

**Models Tested:** - Logistic Regression - Decision Tree Classifier - Random Forest Classifier

**Features Used:** - `grain_type_encoded` - Type of grain (Rice, Wheat, Maize, etc.) - `total_bags` - Number of bags stored - `total_weight_kg` - Total weight in kilograms - `storage_duration_days` - Days stored - `monthly_rent_per_bag` - Rent charged per bag per month - `total_rent_paid` - Cumulative rent paid - `activity_status_encoded` - Current activity status - `sold_status_encoded` - Sale status

**Target Variable:** `price_category` (Low Price, Medium Price, High Price)

**Evaluation Metrics:** - **Accuracy:** 85.2% (best model) - **Precision:** 0.84 - **Recall:** 0.85 - **F1-Score:** 0.84 - **Confusion Matrix:** 3x3 matrix showing predictions

**Best Model:** Random Forest Classifier (selected based on F1-Score)

## 9.3 Model 2: Profit/Loss Classification

**File:** `wms-analytics/profit_classification.ipynb`

**Objective:** Predict whether a customer will make profit or loss

**Models Tested:** - Logistic Regression - Decision Tree Classifier - Random Forest Classifier

**Features Used:** - `grain_type_encoded` - `total_bags` - `total_weight_kg` - `storage_duration_days` - `monthly_rent_per_bag` - `total_rent_paid` - `sale_price_per_kg`

**Target Variable:** `is_profitable` (0 = Loss, 1 = Profit)

**Evaluation Metrics:** - **Accuracy:** 92.1% (best model) - **Precision:** 0.91 - **Recall:** 0.93 - **F1-Score:** 0.92 - **Confusion Matrix:** 2x2 matrix

**Best Model:** Random Forest Classifier

## 9.4 Model 3: Storage Duration Prediction

**File:** `wms-analytics/storage_duration.ipynb`

**Objective:** Predict storage duration category

**Models Tested:** - Logistic Regression - Decision Tree Classifier - Random Forest Classifier

**Features Used:** - `grain_type_encoded` - `total_bags` - `total_weight_kg` - `monthly_rent_per_bag` - `activity_status_encoded`

**Target Variable:** `duration_category` (Short-term, Medium-term, Long-term)

**Evaluation Metrics:** - **Accuracy:** 78.5% (best model) - **Precision:** 0.77 - **Recall:** 0.79 - **F1-Score:** 0.78 - **Confusion Matrix:** 3x3 matrix

**Best Model:** Random Forest Classifier

## 9.5 Dataset Information

**Data Source:** `wms-analytics/CUSTOMER_ACTIVITIES.csv` & `GRAIN_MOVEMENTS.csv`

**Dataset Creation Process:**

```
import pandas as pd
import numpy as np
from datetime import datetime, timedelta

# Generate synthetic customer activities
customers = []
for i in range(1, 501):
    customer = {
        'customer_id': f'CUST{i:04d}',
        'grain_type': np.random.choice(['Rice', 'Wheat', 'Maize', 'Baj
        'total_bags': np.random.randint(50, 500),
        'storage_duration_days': np.random.randint(15, 180),
        # ... more fields
    }
    customers.append(customer)

df = pd.DataFrame(customers)
df.to_csv('CUSTOMER_ACTIVITIES.csv', index=False)
```

**Dataset Size:** 500 customer records with 15+ features

## 9.6 Model Deployment

**Saved Models:** - `model1_price_prediction_BEST.pkl` - `model2_profit_classification_BEST.pkl` - `model3_duration_prediction_BEST.pkl`

**Loading Models in Backend:**

```
// server/services/mlService.js
const { spawn } = require('child_process');

async function predictPriceCategory(features) {
  return new Promise((resolve, reject) => {
    const python = spawn('python', [
      'wms-analytics/predict.py',
      'model1',
      JSON.stringify(features)
    ]);

    let result = '';
    python.stdout.on('data', (data) => {
      result += data.toString();
    });

    python.on('close', (code) => {
      if (code === 0) {
        resolve(JSON.parse(result));
      } else {
        reject(new Error('Prediction failed'));
      }
    });
  });
}
```

---

# 10. Workflows & Business Processes

## 10.1 Customer Onboarding Workflow

1. **Registration** - Customer submits application online
2. **Verification** - Admin verifies documents
3. **Approval** - Warehouse owner approves customer
4. **Storage Allocation** - Assign storage location
5. **Contract Generation** - Digital contract created
6. **Payment Setup** - Configure payment terms
7. **Onboarding Complete** - Customer gets access

## 10.2 Monthly Billing Process

1. **Calculation** - System calculates monthly rent
2. **Invoice Generation** - PDF invoice created
3. **Email Notification** - Invoice sent to customer
4. **Payment Due** - 7-day payment window
5. **Payment Collection** - Customer pays online/offline
6. **Receipt Generation** - Payment receipt issued
7. **Accounting Update** - Financial records updated

## 10.3 Weighbridge Operations

1. **Vehicle Entry** - Driver checks in
2. **Empty Weight** - Vehicle weighed empty
3. **Loading** - Grain loaded onto vehicle
4. **Loaded Weight** - Vehicle weighed again
5. **Net Weight Calculation** - Automatic calculation
6. **Slip Generation** - QR-coded weighbridge slip
7. **Vehicle Exit** - Driver checks out

## 10.4 Quality Inspection

1. **Request** - Customer requests inspection

2. **Scheduling** - Worker assigned and scheduled
3. **Inspection** - Physical grain inspection
4. **Data Entry** - Parameters recorded
5. **Report Generation** - Quality certificate created
6. **Notification** - Customer notified of results
7. **Storage** - Report saved to document vault

---

# 11. API Documentation

## 11.1 Authentication APIs

**POST /api/auth/register** - Register new customer - Body: `{ name, email, password, mobile, role }` - Response: `{ success, message, userId }`

**POST /api/auth/login** - User login with language preference - Body: `{ email, password, language }` - Response: `{ token, user: { id, name, role, preferred_language } }`

**GET /api/auth/me** - Get current user details - Headers: `Authorization: Bearer <token>` - Response: `{ user: { ... } }`

## 11.2 Customer APIs

**GET /api/customers** - Get all customers (Owner only) - Query params: `?status=active&page=1&limit=10` - Response: `{ customers: [...], total, page, pages }`

**POST /api/customers** - Create new customer - Body: `{ name, email, mobile, address, ... }` - Response: `{ success, customer: { ... } }`

**PUT /api/customers/:id** - Update customer details - Body: `{ name, email, ... }` - Response: `{ success, customer: { ... } }`

**DELETE /api/customers/:id** - Delete customer - Response: `{ success, message }`

## 11.3 Analytics APIs

**GET /api/analytics/grain-distribution** - Get grain type distribution - Response: `{ grains: [...], total_bags, total_weight_kg }`

**GET /api/analytics/storage-duration** - Get storage duration analytics - Response: `{ categories: {...}, average_duration_days, trends: [...] }`

**GET /api/analytics/customer-flow** - Get customer in/out flow - Response: `{ monthly_data: [...], churn_rate, retention_rate }`

**POST /api/analytics/ml-predict** - ML model predictions - Body: `{ model, customer_id, features: {...} }` - Response: `{ prediction, probability, confidence, recommendation }`

## 11.4 Payment APIs

**GET /api/payments** - Get payment history - Query: `?customer_id=...&status=pending` - Response: `{ payments: [...], total_amount }`

**POST /api/payments** - Record new payment - Body: `{ customer_id, amount, method, transaction_id }` - Response: `{ success, payment: {...}, receipt_url }`

**POST /api/payments/razorpay/create-order** - Create Razorpay order - Body: `{ amount }` - Response: `{ order_id, amount, currency }`

---

# 12. Database Schema

## 12.1 User Collection

```
{
  _id: ObjectId,
  name: String,
  email: String (unique),
```

```
  password: String (hashed),
  mobile: String,
  role: String (owner/customer/worker),
  preferred_language: String (en/te),
  created_at: Date,
  updated_at: Date
}
```

## 12.2 Storage Allocation Collection

```
{
  _id: ObjectId,
  customer_id: ObjectId (ref: User),
  grain_type: String,
  total_bags: Number,
  total_weight_kg: Number,
  storage_location: String,
  storage_start_date: Date,
  storage_end_date: Date,
  storage_duration_days: Number,
  monthly_rent_per_bag: Number,
  total_rent_paid: Number,
  status: String (active/completed),
  created_at: Date
}
```

## 12.3 Transaction Collection

```
{
  _id: ObjectId,
  customer_id: ObjectId (ref: User),
  transaction_type: String (payment/rent/loan),
  amount: Number,
  payment_method: String,
  payment_status: String (pending/completed/failed),
  due_date: Date,
  payment_date: Date,
  transaction_id: String,
  receipt_url: String,
```

```
  created_at: Date
}
```

### 12.4 Vehicle Collection

```
{
  _id: ObjectId,
  vehicle_number: String (unique),
  vehicle_type: String,
  driver_name: String,
  driver_mobile: String,
  empty_weight: Number,
  loaded_weight: Number,
  net_weight: Number,
  entry_time: Date,
  exit_time: Date,
  qr_code: String,
  created_at: Date
}
```

---

## 13. Security & Authentication

### 13.1 Authentication Strategy

- JWT Tokens - 7-day expiry
- bcryptjs - Password hashing (10 salt rounds)
- HTTP-only Cookies - Token storage
- Role-based Authorization - Middleware protection

### 13.2 Security Features

- Input validation using express-validator
- MongoDB injection prevention
- XSS protection with helmet.js

- CORS configuration
- Rate limiting (100 requests/15 minutes)
- File upload restrictions (size, type)

---

# 14. Payment Integration

## 14.1 Razorpay Setup

```
const Razorpay = require('razorpay');

const razorpay = new Razorpay({
  key_id: process.env.RAZORPAY_KEY_ID,
  key_secret: process.env.RAZORPAY_KEY_SECRET
});
```

## 14.2 Payment Flow

1. Customer initiates payment
2. Backend creates Razorpay order
3. Frontend displays Razorpay checkout
4. Customer completes payment
5. Webhook verifies payment
6. Database updated
7. Receipt generated and emailed

---

# 15. Real-time Features

## 15.1 Socket.IO Implementation

```
// server/server.js
const io = require('socket.io')(server, {
  cors: { origin: 'http://localhost:3000' }
});

io.on('connection', (socket) => {
  console.log('Client connected:', socket.id);

  socket.on('join-room', (userId) => {
    socket.join(`user-${userId}`);
  });

  socket.on('disconnect', () => {
    console.log('Client disconnected');
  });
});

// Emit events
io.to(`user-${userId}`).emit('payment-received', paymentData);
```

### 15.2 Real-time Events

- Payment confirmations
- Storage updates
- Task assignments
- Alert notifications
- Queue updates

---

## 16. Deployment Guide

### 16.1 Environment Variables

```
# Server
PORT=5000
NODE_ENV=production
```

```
JWT_SECRET=your-secret-key

# Database
MONGODB_URI=mongodb://localhost:27017/wms

# Razorpay
RAZORPAY_KEY_ID=rzp_...
RAZORPAY_KEY_SECRET=...

# Email
SMTP_HOST=smtp.gmail.com
SMTP_PORT=587
SMTP_USER=your-email@gmail.com
SMTP_PASS=app-password

# AI
OPENAI_API_KEY=sk-...
CLAUDE_API_KEY=sk-ant-...

# SMS
TWILIO_ACCOUNT_SID=AC...
TWILIO_AUTH_TOKEN=...
```

## 16.2 Build Commands

```
# Install dependencies
npm install
cd client && npm install

# Build frontend
cd client && npm run build

# Start production server
npm start
```

## 16.3 Deployment Platforms

- **Frontend:** Vercel, Netlify
- **Backend:** Heroku, AWS EC2, DigitalOcean

- **Database:** MongoDB Atlas
- **File Storage:** AWS S3, Cloudinary

---

# Appendix

## A. Technology Versions

- Node.js: 18.x or higher
- npm: 9.x or higher
- MongoDB: 5.x or higher
- Python: 3.9 or higher

## B. External Services

- Razorpay Payment Gateway
- OpenAI / Claude API
- Twilio SMS Service
- Gmail SMTP / SendGrid

## C. Support & Maintenance

- Regular security updates
- Monthly ML model retraining
- Database backups (daily)
- Performance monitoring

---

**Document Version:** 2.0
**Last Updated:** January 27, 2026
**Contact:** support@wms.com
**Website:** www.wms.com

---

# Change Log

**Version 2.0 (January 27, 2026):** - ✅ Added enhanced Analytics & Visualization Module - ✅ Integrated ML models with classification metrics - ✅ Implemented AI-Powered Alert System with node-cron - ✅ Added Multi-Language Support (i18n) - English & Telugu - ✅ Enhanced security features - ✅ Updated API documentation

**Version 1.0 (January 2026):** - Initial project documentation - Core features documented - Basic workflow descriptions

---

**End of Documentation**