Wild Octave Organics Invoice Processing System - Complete Documentation

Table of Contents

Project Overview

The Wild Octave Organics Invoice Processing System is a comprehensive web application designed to streamline the invoice processing workflow for health food businesses. It provides automated invoice processing, intelligent product categorization, Square POS integration, and real-time inventory management.

Key Features

- **Automated Invoice Processing**: Upload PDF invoices and extract line items automatically
- **Square API Integration**: Bi-directional sync with Square POS system
- **Intelligent Product Linking**: AI-powered matching of invoice items to Square products
- **Real-time Inventory Management**: Track stock levels and receive notifications
- **Automated Daily Sync**: Scheduled synchronization at 5am AEST
- **Webhook Integration**: Real-time updates from Square
- **Category Management**: Automated product categorization with custom markup
- **Admin Dashboard**: Complete control over Square integration settings
- **Stock Tracking**: Monitor received stock and inventory levels

Business Benefits

- Reduces manual data entry by 90%
- Ensures accurate inventory tracking
- Automates pricing calculations with markup
- Provides real-time business insights
- Streamlines vendor invoice processing
- Maintains synchronized POS system

Technology Stack & Architecture

Frontend Stack

- **Next.js 14**: React framework with App Router
- **React 18**: Component library with hooks
- **TypeScript**: Type-safe JavaScript
- **Tailwind CSS**: Utility-first CSS framework
- **Radix UI**: Accessible component primitives
- **Shadcn/ui**: Pre-built component library
- **Framer Motion**: Animation library

Backend Stack

- **Next.js API Routes**: Server-side API endpoints
- **Prisma ORM**: Database toolkit and query builder
- **PostgreSQL**: Primary database
- **NextAuth.js**: Authentication framework
- **Square API v2**: POS system integration

Data Processing

- **React Dropzone**: File upload handling
- **CSV Parser**: Invoice data extraction
- **Plotly.js**: Data visualization
- **Chart.js**: Additional charting capabilities

Infrastructure

- **Vercel**: Deployment platform (recommended)
- **AWS/Azure**: Alternative deployment options
- **Docker**: Containerization support
- **GitHub Actions**: CI/CD pipeline

Architecture Overview

```
┌─────────────────┐    ┌─────────────────┐    ┌─────────────────┐
│   Frontend      │    │   Backend       │    │   External      │
│   (Next.js)     │◄──►│   (API Routes)  │◄──►│   (Square API)  │
└─────────────────┘    └─────────────────┘    └─────────────────┘
         │                      │                      │
         ▼                      ▼                      ▼
┌─────────────────┐    ┌─────────────────┐    ┌─────────────────┐
│   UI Components │    │   Database      │    │   Webhooks      │
│   (React/TS)    │    │   (PostgreSQL)  │    │   (Real-time)   │
└─────────────────┘    └─────────────────┘    └─────────────────┘
```

Setup & Installation

Prerequisites

- Node.js 18+ and npm
- PostgreSQL database
- Square Developer Account

- Git

Step-by-Step Installation

1. **Clone the Repository**
   `bash`
   `git clone <repository-url>`
   `cd wild-octave-organics`

2. **Install Dependencies**
   `bash`
   `npm install`

3. **Environment Configuration**
   `bash`
   `cp .env.example .env`
   Edit `.env` with your configuration (see Environment Configuration section)

4. **Database Setup**
   ```bash
   # Generate Prisma client
   npx prisma generate

# Push schema to database
npx prisma db push

# Seed initial data
npx prisma db seed
```

1. **Verify Installation**
   `bash`
   `npm run build`
   `npm run dev`

2. **Access Application**
   - Main Application: http://localhost:3000
   - Square Admin Dashboard: http://localhost:3000/square
   - Test Dashboard: http://localhost:3000/test-dashboard

Development Scripts

```
# Development server
npm run dev

# Production build
npm run build

# Start production server
npm run start

# Lint code
npm run lint

# Database operations
npx prisma studio          # Database GUI
npx prisma db push         # Push schema changes
npx prisma db seed         # Seed database
npx prisma generate        # Generate client
```

Environment Configuration

Required Environment Variables

Create a `.env` file in the root directory with the following variables:

```
# Database Configuration
DATABASE_URL="postgresql://username:password@localhost:5432/wild_octave_db"

# Square API Configuration
SQUARE_APPLICATION_ID="your_square_application_id"
SQUARE_ACCESS_TOKEN="your_square_access_token"
SQUARE_ENVIRONMENT="sandbox"  # or "production"
SQUARE_WEBHOOK_SIGNATURE_KEY="your_webhook_signature_key"
SQUARE_LOCATION_ID="your_square_location_id"

# NextAuth Configuration
NEXTAUTH_URL="http://localhost:3000"
NEXTAUTH_SECRET="your_nextauth_secret"

# Application Configuration
NODE_ENV="development"
NEXT_PUBLIC_APP_URL="http://localhost:3000"

# Cron Job Configuration
CRON_SECRET="your_cron_secret_key"
TIMEZONE="Australia/Sydney"

# Optional: Email Configuration
SMTP_HOST="your_smtp_host"
SMTP_PORT="587"
SMTP_USER="your_smtp_user"
SMTP_PASS="your_smtp_password"
```

Environment-Specific Configurations

Development Environment

```
SQUARE_ENVIRONMENT="sandbox"
DATABASE_URL="postgresql://localhost:5432/wild_octave_dev"
NEXTAUTH_URL="http://localhost:3000"
```

Production Environment

```
SQUARE_ENVIRONMENT="production"
DATABASE_URL="your_production_database_url"
NEXTAUTH_URL="https://your-domain.com"
```

Square API Setup

1. **Create Square Developer Account**
   - Visit https://developer.squareup.com/
   - Create a new application
   - Note your Application ID

2. **Generate Access Token**
   - In Square Dashboard, go to Credentials
   - Generate Sandbox/Production Access Token
   - Copy the token to your `.env` file

3. **Configure Webhooks**
   - Set webhook URL: `https://your-domain.com/api/square/webhooks`
   - Subscribe to events: `inventory.count.updated`, `catalog.version.updated`
   - Copy signature key to `.env` file

4. **Get Location ID**
   - Use Square API or Dashboard to find your location ID
   - Add to `.env` file

Square API Integration

Overview

The application integrates with Square API v2 to provide:
- Real-time inventory synchronization
- Product catalog management
- Webhook event processing
- Daily automated sync

Integration Components

1. Square API Client (`lib/square-api-client.ts`)

```
// Handles all Square API communications
export class SquareApiClient {
  private client: Client;

  constructor() {
    this.client = new Client({
      accessToken: process.env.SQUARE_ACCESS_TOKEN,
      environment: process.env.SQUARE_ENVIRONMENT === 'production'
        ? Environment.Production
        : Environment.Sandbox
    });
  }

  // Methods for catalog, inventory, and webhook operations
}
```

2. Sync Service (`lib/square-sync.ts`)

```
// Manages synchronization operations
export class SquareSync {
  async syncProducts(): Promise<SyncResult>
  async syncInventory(): Promise<SyncResult>
  async processWebhookEvent(event: WebhookEvent): Promise<void>
}
```

3. Webhook Handler (`app/api/square/webhooks/route.ts`)

```
// Processes real-time Square events
export async function POST(request: Request) {
  // Verify webhook signature
  // Process event based on type
  // Update local database
}
```

API Endpoints

Product Management

- `GET /api/square/products` - List all Square products
- `POST /api/square/sync` - Manual sync trigger
- `GET /api/square/inventory` - Get inventory levels

Product Linking

- `GET /api/square/product-links` - List product links
- `POST /api/square/product-links` - Create product link
- `GET /api/square/product-links/suggestions` - Get link suggestions

Stock Management

- `POST /api/square/receive-stock` - Mark stock as received
- `GET /api/square/inventory/:locationId` - Get location inventory

Daily Sync Process

The system performs automated daily synchronization at 5am AEST:

1. **Product Sync**: Updates product catalog from Square
2. **Inventory Sync**: Synchronizes inventory levels
3. **Link Validation**: Validates existing product links
4. **Error Reporting**: Logs any sync issues

Cron Job Setup

```
// lib/cron-scheduler.ts
export function scheduleDailySync() {
  cron.schedule('0 5 * * *', async () => {
    await performDailySync();
  }, {
    timezone: 'Australia/Sydney'
  });
}
```

Webhook Events

The system handles the following Square webhook events:

Inventory Updates

```
// When inventory changes in Square
{
  "type": "inventory.count.updated",
  "data": {
    "object": {
      "inventory_counts": [...]
    }
  }
}
```

Catalog Updates

```
// When products are modified in Square
{
  "type": "catalog.version.updated",
  "data": {
    "object": {
      "updated_at": "2025-08-04T10:00:00Z"
    }
  }
}
```

Database Schema & Migrations

Schema Overview

The database uses PostgreSQL with Prisma ORM. Key entities include:

Core Entities

- **Invoice**: Vendor invoices with line items

- **LineItem**: Individual products from invoices
- **Product**: Master product catalog
- **Category**: Product categories with markup
- **Vendor**: Invoice suppliers

Square Integration Entities

- **SquareProduct**: Products from Square catalog
- **SquareInventory**: Inventory levels by location
- **ProductLink**: Links between invoice items and Square products
- **WebhookEvent**: Square webhook event log
- **SyncLog**: Synchronization operation log

Key Relationships

```
-- Invoice to LineItems (One-to-Many)
Invoice ||--o{ LineItem

-- LineItem to SquareProduct (Many-to-One)
LineItem }o--|| SquareProduct

-- SquareProduct to SquareInventory (One-to-Many)
SquareProduct ||--o{ SquareInventory

-- Product to Category (Many-to-One via mapping)
Product }o--|| Category
```

Database Migrations

Initial Setup

```
# Generate Prisma client
npx prisma generate

# Push schema to database
npx prisma db push

# Seed initial data
npx prisma db seed
```

Schema Updates

```
# After modifying schema.prisma
npx prisma db push

# Generate new client
npx prisma generate

# Reset database (development only)
npx prisma db reset
```

Backup and Restore

```
# Backup database
pg_dump wild_octave_db > backup.sql

# Restore database
psql wild_octave_db < backup.sql
```

Seed Data

The seed script (`scripts/seed.ts`) creates:
- Default categories with markup percentages
- Sample vendors
- Test products for development

```
// Example seed data
const categories = [
  { name: 'Supplements', markup: 0.40 },
  { name: 'Organic Foods', markup: 0.35 },
  { name: 'Personal Care', markup: 0.45 }
];
```

Deployment Guide

Platform Options

1. Vercel (Recommended)

```
# Install Vercel CLI
npm i -g vercel

# Deploy
vercel

# Set environment variables
vercel env add DATABASE_URL
vercel env add SQUARE_ACCESS_TOKEN
# ... add all required env vars
```

2. AWS Deployment

Using AWS Amplify:

```
# Install Amplify CLI
npm install -g @aws-amplify/cli

# Initialize project
amplify init

# Add hosting
amplify add hosting

# Deploy
amplify publish
```

Using AWS ECS with Docker:

```
# Dockerfile
FROM node:18-alpine

WORKDIR /app
COPY package*.json ./
RUN npm ci --only=production

COPY . .
RUN npm run build

EXPOSE 3000
CMD ["npm", "start"]
```

3. Azure Deployment

Using Azure Static Web Apps:

```
# .github/workflows/azure-static-web-apps.yml
name: Azure Static Web Apps CI/CD

on:
  push:
    branches: [ main ]

jobs:
  build_and_deploy_job:
    runs-on: ubuntu-latest
    steps:
    - uses: actions/checkout@v2
    - name: Build And Deploy
      uses: Azure/static-web-apps-deploy@v1
      with:
        azure_static_web_apps_api_token: ${{ secrets.AZURE_STATIC_WEB_APPS_API_TOKEN }}
        repo_token: ${{ secrets.GITHUB_TOKEN }}
        action: "upload"
        app_location: "/"
        api_location: "api"
        output_location: "out"
```

4. Docker Deployment

```yaml
# docker-compose.yml
version: '3.8'
services:
  app:
    build: .
    ports:
      - "3000:3000"
    environment:
      - DATABASE_URL=postgresql://postgres:password@db:5432/wild_octave
      - SQUARE_ACCESS_TOKEN=${SQUARE_ACCESS_TOKEN}
    depends_on:
      - db

  db:
    image: postgres:15
    environment:
      - POSTGRES_DB=wild_octave
      - POSTGRES_PASSWORD=password
    volumes:
      - postgres_data:/var/lib/postgresql/data

volumes:
  postgres_data:
```

Pre-Deployment Checklist

- [ ] Environment variables configured
- [ ] Database migrations applied
- [ ] Square API credentials valid
- [ ] Webhook endpoints accessible
- [ ] SSL certificate configured
- [ ] Domain DNS configured
- [ ] Backup strategy implemented

Post-Deployment Verification

1. **Health Check**
   `bash`
   ```
   curl https://your-domain.com/api/health
   ```

2. **Database Connection**
   `bash`
   ```
   npx prisma db push --preview-feature
   ```

3. **Square Integration**
   `bash`
   ```
   curl https://your-domain.com/api/square/products
   ```

4. **Webhook Testing**
   `bash`
   ```
   # Test webhook endpoint
   curl -X POST https://your-domain.com/api/square/webhooks \
     -H "Content-Type: application/json" \
     -d '{"test": true}'
   ```

API Endpoints Documentation

Authentication

Most endpoints require authentication. Include the session token in requests:

```
// Client-side API call
const response = await fetch('/api/invoices', {
  headers: {
    'Content-Type': 'application/json',
    'Authorization': `Bearer ${sessionToken}`
  }
});
```

Invoice Management

Upload Invoice

```
POST /api/process-invoice
Content-Type: multipart/form-data

{
  "file": <PDF file>,
  "vendor": "Vendor Name"
}
```

Response:

```
{
  "success": true,
  "invoiceId": "clx123...",
  "lineItems": [
    {
      "productName": "Organic Almonds",
      "quantity": 10,
      "unitPrice": 15.50,
      "totalPrice": 155.00
    }
  ]
}
```

Get Invoices

```
GET /api/invoices?page=1&limit=10&vendor=VendorName
```

Response:

```json
{
  "invoices": [...],
  "pagination": {
    "page": 1,
    "limit": 10,
    "total": 50,
    "pages": 5
  }
}
```

Square Integration

Sync Products

```
POST /api/square/sync
Content-Type: application/json

{
  "syncType": "products" | "inventory" | "all"
}
```

Get Square Products

```
GET /api/square/products?search=organic&category=supplements
```

Create Product Link

```
POST /api/square/product-links
Content-Type: application/json

{
  "invoiceProductName": "Organic Almonds 1kg",
  "squareProductId": "square_product_id",
  "confidence": 0.95,
  "isManualLink": true
}
```

Get Link Suggestions

```
GET /api/square/product-links/suggestions?productName=Organic%20Almonds
```

Response:

```json
{
  "suggestions": [
    {
      "squareProduct": {
        "id": "square_id",
        "name": "Organic Raw Almonds",
        "price": 15.50
      },
      "confidence": 0.95,
      "reason": "Exact name match"
    }
  ]
}
```

Category Management

Get Categories

```
GET /api/categories
```

Assign Category

```
POST /api/assign-category
Content-Type: application/json

{
  "lineItemId": "clx123...",
  "categoryId": "cat123..."
}
```

Inventory Management

Receive Stock

```
POST /api/square/receive-stock
Content-Type: application/json

{
  "lineItemId": "clx123...",
  "quantityReceived": 10,
  "notes": "Received in good condition"
}
```

Get Inventory

```
GET /api/square/inventory?locationId=location123
```

Webhook Endpoints

Square Webhooks

```
POST /api/square/webhooks
Content-Type: application/json
X-Square-Signature: <signature>

{
  "merchant_id": "merchant123",
  "type": "inventory.count.updated",
  "event_id": "event123",
  "created_at": "2025-08-04T10:00:00Z",
  "data": {
    "type": "inventory",
    "id": "inventory123",
    "object": {...}
  }
}
```

Error Responses

All endpoints return consistent error responses:

```json
{
  "error": true,
  "message": "Detailed error message",
  "code": "ERROR_CODE",
  "details": {
    "field": "validation error details"
  }
}
```

Common HTTP status codes:
- `200` - Success
- `400` - Bad Request
- `401` - Unauthorized
- `403` - Forbidden
- `404` - Not Found
- `500` - Internal Server Error

Feature Usage Instructions

1. Invoice Processing

Uploading Invoices

1. Navigate to the main dashboard
2. Click “Upload Invoice” or drag & drop PDF files
3. Select the vendor from the dropdown
4. Click “Process Invoice”
5. Review extracted line items
6. Assign categories to products
7. Link products to Square inventory (optional)

Managing Line Items

1. View processed invoices in the dashboard
2. Click on any line item to edit details
3. Assign categories for automatic markup calculation
4. Link to Square products for inventory tracking
5. Mark items as “needs clarification” if required

2. Square Integration

Initial Setup

1. Navigate to `/square` admin dashboard
2. Enter Square API credentials
3. Click “Test Connection” to verify
4. Run initial sync to import products
5. Configure webhook URL in Square Dashboard

Product Linking

1. Go to the Product Linking interface
2. View unlinked invoice items
3. Click “Find Matches” for automatic suggestions
4. Review and approve suggested links
5. Create manual links for items without matches

Daily Sync Management

1. View sync logs in the admin dashboard
2. Monitor sync status and errors
3. Manually trigger sync if needed
4. Configure sync schedule (default: 5am AEST)

3. Inventory Management

Stock Receiving

1. Process invoices as usual
2. When stock arrives, mark items as “received”
3. System automatically updates Square inventory
4. View inventory levels in the dashboard

Inventory Tracking

1. Monitor stock levels in real-time
2. Receive low stock alerts
3. View inventory history and movements
4. Generate inventory reports

4. Category Management

Setting Up Categories

1. Navigate to Categories section
2. Create categories with markup percentages
3. Assign categories to products
4. System automatically calculates final prices

Markup Configuration

1. Set different markup percentages per category
2. Override markup for specific products
3. Apply bulk category assignments
4. Monitor profit margins

5. Reporting and Analytics

Invoice Reports

1. View invoice processing statistics
2. Monitor vendor performance
3. Track processing times
4. Export data to CSV

Inventory Reports

1. Stock level reports
2. Movement history
3. Low stock alerts
4. Turnover analysis

6. Admin Functions

User Management

1. Add/remove users
2. Set user permissions
3. Monitor user activity
4. Configure access levels

System Configuration

1. Configure Square API settings
2. Set up webhook endpoints
3. Manage sync schedules
4. Configure email notifications

Troubleshooting Guide

Common Issues and Solutions

1. Square API Connection Issues

Problem: "Failed to connect to Square API"

Diagnosis:

```
# Test API connection
curl -H "Authorization: Bearer YOUR_ACCESS_TOKEN" \
  https://connect.squareupsandbox.com/v2/locations
```

Solutions:
- Verify access token is correct and not expired
- Check if using correct environment (sandbox vs production)

- Ensure Square application has required permissions
- Check network connectivity and firewall settings

2. Database Connection Issues

Problem: “Database connection failed”

Diagnosis:

```
# Test database connection
npx prisma db push --preview-feature
```

Solutions:
- Verify DATABASE_URL format: `postgresql://user:pass@host:port/db`
- Check database server is running
- Verify credentials and permissions
- Test network connectivity to database

3. Invoice Processing Failures

Problem: “Failed to process invoice PDF”

Diagnosis:
- Check file format (must be PDF)
- Verify file size (max 10MB)
- Check PDF is not password protected
- Ensure PDF contains extractable text

Solutions:
- Convert image-based PDFs to text-based
- Reduce file size if too large
- Remove password protection
- Use OCR tools for scanned documents

4. Webhook Not Receiving Events

Problem: “Square webhooks not working”

Diagnosis:

```
# Check webhook endpoint
curl -X POST https://your-domain.com/api/square/webhooks \
  -H "Content-Type: application/json" \
  -d '{"test": true}'
```

Solutions:
- Verify webhook URL is publicly accessible
- Check webhook signature validation
- Ensure HTTPS is configured correctly
- Verify webhook events are subscribed in Square Dashboard

5. Product Linking Issues

Problem: “Product suggestions not appearing”

Solutions:
- Run manual Square sync to update product catalog
- Check product names for special characters
- Verify Square products are active
- Clear cache and refresh page

6. Daily Sync Failures

Problem: "Daily sync not running"

Diagnosis:

```
# Check cron job logs
tail -f /var/log/cron.log
```

Solutions:
- Verify cron job is configured correctly
- Check server timezone settings
- Ensure application is running continuously
- Review sync logs for specific errors

7. Performance Issues

Problem: "Application running slowly"

Diagnosis:
- Check database query performance
- Monitor server resources (CPU, memory)
- Review network latency
- Check for memory leaks

Solutions:
- Optimize database queries
- Add database indexes
- Increase server resources
- Implement caching strategies

8. Authentication Issues

Problem: "Login not working"

Solutions:
- Verify NextAuth configuration
- Check session storage (cookies/database)
- Ensure NEXTAUTH_SECRET is set
- Clear browser cookies and cache

Debug Mode

Enable debug mode for detailed logging:

```
# Add to .env
DEBUG=true
LOG_LEVEL=debug
```

Log Files

Check application logs:

```
# Development logs
tail -f dev-server.log

# Production logs
tail -f production-server.log

# Database logs
tail -f /var/log/postgresql/postgresql.log
```

Performance Monitoring

Monitor application performance:

```
# Check memory usage
ps aux | grep node

# Monitor database connections
SELECT * FROM pg_stat_activity;

# Check disk space
df -h
```

Getting Help

1. **Check Documentation**: Review this guide and README
2. **Search Issues**: Look for similar problems in logs
3. **Test in Isolation**: Isolate the problem component
4. **Check Dependencies**: Ensure all packages are up to date
5. **Contact Support**: Provide detailed error messages and logs

Maintenance Tasks

Weekly Tasks

- [ ] Review sync logs for errors
- [ ] Check database performance
- [ ] Monitor disk space usage
- [ ] Review security logs

Monthly Tasks

- [ ] Update dependencies
- [ ] Review and archive old invoices
- [ ] Check backup integrity
- [ ] Performance optimization review

Quarterly Tasks

- [ ] Security audit
- [ ] Database optimization
- [ ] Disaster recovery testing
- [ ] User access review

Document Version: 1.0.0
Last Updated: August 4, 2025
Maintained By: Wild Octave Organics Development Team