BRD Distribution - Delivery Manifest System Architecture Documentation

Document Version: 1.0

Generated: February 8, 2026

System: Delivery Manifest System v2.0

Table of Contents

1. System Overview

1.1 Introduction

The BRD Distribution Delivery Manifest System is a comprehensive web-based application designed to automate and manage the delivery manifest process for distribution operations. The system handles invoice processing, manifest creation, dispatch reporting, and tracking of delivery operations.

1.2 Purpose

- Invoice Processing: Automatically detect and process PDF invoices from input folders
- Manifest Management: Create, manage, and track delivery manifests
- Dispatch Reporting: Generate detailed dispatch reports with driver, vehicle, and cargo information
- Order Tracking: Track orders from processing to delivery
- Credit Note Handling: Support for partial and full credit notes

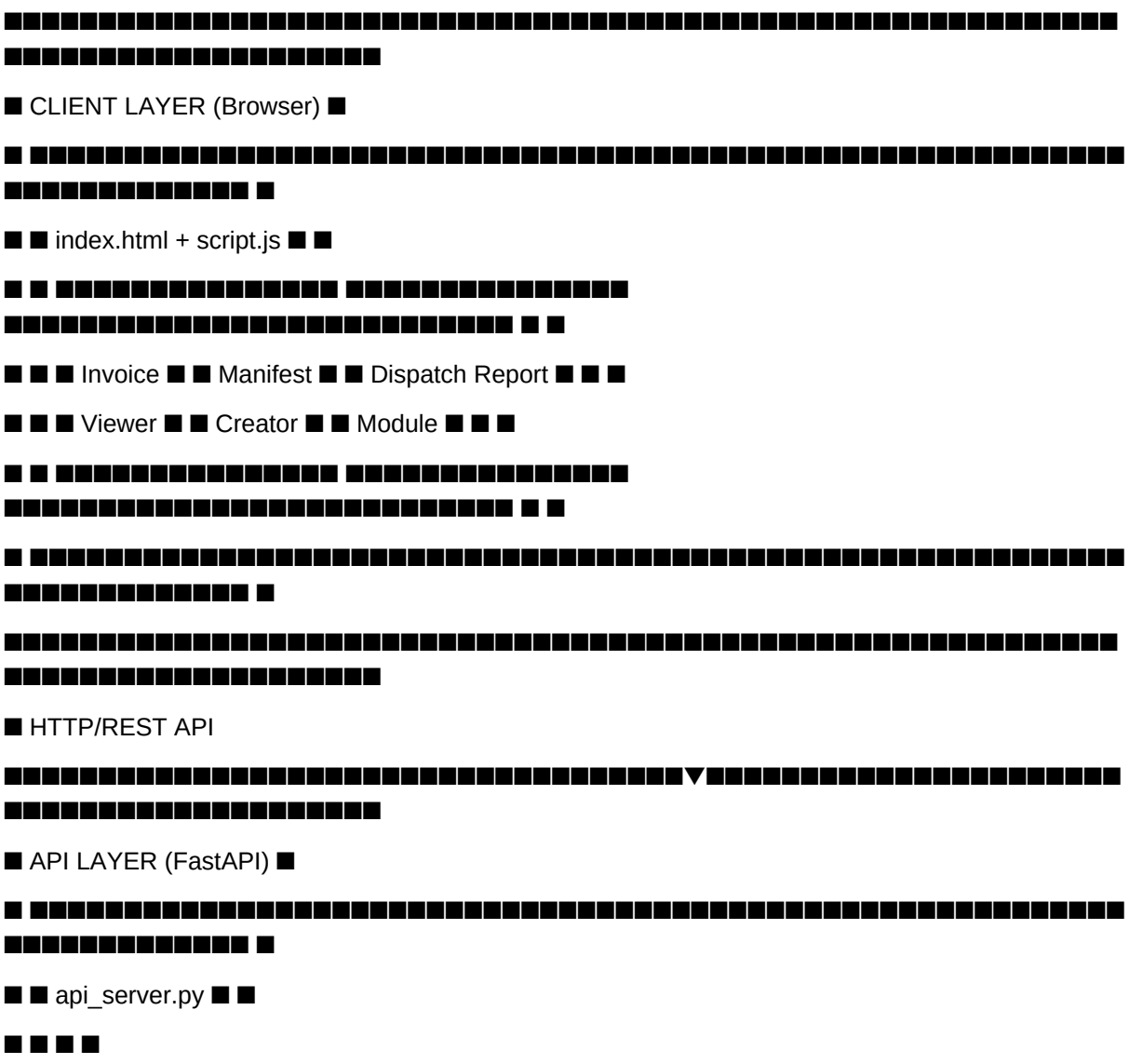
1.3 Technology Stack

1.4 System Location

- Root Directory: C:\Users\Assault\OneDrive\Documents\Delivery Route
- Database: delivery.db
- API Server: api_server.py (FastAPI)
- Frontend: index.html + script.js

2. Architecture Diagram

2.1 High-Level Architecture



■ ■ ■■■■■■■■■■■■■■■■■ ■■■■■■■■■■■■■■■■■
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■ ■ ■

■ ■ ■ Endpoints ■ ■ Auth/CORS ■ ■ File Watcher Thread ■ ■ ■

■ ■ ■ (REST) ■ ■ Middleware ■ ■ (Background) ■ ■ ■

■ ■ ■■■■■■■■■■■■■■■■■ ■■■■■■■■■■■■■■■■■
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■ ■ ■

■ ■■
■■■■■■■■■■■■■■■ ■

■■
■■■■■■■■■■■■■■■■■■■■■■■

■

■■■▼■■■■■■■■■■■■■■■■■■■■■■■■■■■
■■■■■■■■■■■■■■■■■■■■■■■

■ BUSINESS LOGIC LAYER ■

■ ■■
■■■■■■■■■■■■■■■ ■

■ ■ database.py (SQLite Access) ■ ■

■ ■ ■ ■

■ ■ ■■■■■■■■■■■■■■■■■ ■■■■■■■■■■■■■■■■■
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■ ■ ■

■ ■ ■ Orders ■ ■ Manifests ■ ■ Reports/Stats ■ ■ ■

■ ■ ■ Functions ■ ■ Functions ■ ■ Module ■ ■ ■

■ ■ ■■■■■■■■■■■■■■■■■ ■■■■■■■■■■■■■■■■■
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■ ■ ■

■ ■■
■■■■■■■■■■■■■■■ ■

■■
■■■■■■■■■■■■■■■■■■■■■■■

■

■■■▼■■■■■■■■■■■■■■■■■■■■■■■■■■■
■■■■■■■■■■■■■■■■■■■■■■■

■ DATA STORAGE LAYER ■

■ ■■
■■■■■■■■■■■■■■■ ■

■ ■ delivery.db (SQLite) ■ ■

■ ■ ■ ■

■ ■ ■■■■■■■■■■■■ ■■■■■■■■■■■■ ■■■■■■■■■■■■ ■■■■■■■■■■■■
■■■■■■■■■■■■ ■ ■

■ ■ ■ orders ■ ■ reports ■ ■ users ■ ■ trucks ■ ■ settings ■ ■ ■

■ ■ ■■■■■■■■■■■■ ■■■■■■■■■■■■ ■■■■■■■■■■■■ ■■■■■■■■■■■■
■■■■■■■■■■■■ ■ ■

■ ■■
■■■■■■■■■■■■■ ■

■■■
■■■■■■■■■■■■■■■■■■■■

2.2 Data Flow Architecture

■■■
■■■■■■■■■■■■■■■■■■■■

■ EXTERNAL INPUT SOURCES ■

■ ■

■ ■■■■■■■■■■■■■■■■■■■ ■■■■■■■■■■■■■■■■■■■
■■■■■■■■■■■■■■■■■■■■■■■■■■■ ■

■ ■ PDF Invoices ■ ■ Excel Manifest ■ ■ Manual Input ■ ■

■ ■ (Invoices_ ■ ■ Templates ■ ■ (Web Interface) ■ ■

■ ■ Input/) ■ ■ ■ ■ ■ ■

■ ■■■■■■■■■■■■■■■■■■■ ■■■■■■■■■■■■■■■■■■■
■■■■■■■■■■■■■■■■■■■■■■■■■■■ ■

■■■
■■■■■■■■■■■■■■■■■■■■

■ ■ ■

▼ ▼ ▼

■■■
■■■■■■■■■■■■■■■■■■■■

■ FILE WATCHER SERVICE ■

■ - Polls Invoices_Input folder every 20 seconds ■

■ - Validates file stability (size checks) ■

■ - Triggers invoice_processor.py for new files ■

■■■
■■■■■■■■■■■■■■■■■■

■

▼

■■■
■■■■■■■■■■■■■■■■■■■■

■ INVOICE PROCESSOR ■

■ - Extracts data from PDF invoices using PyPDF2 ■

■ - Parses Excel manifest templates ■

■ - Validates and normalizes data ■

■ - Stores in database via database.py ■

■■■
■■■■■■■■■■■■■■■■■■

■

▼

■■■
■■■■■■■■■■■■■■■■■■■■

■ DATABASE (delivery.db) ■

■ - Stores all orders, manifests, reports, and settings ■

■ - Provides data access via database.py functions ■

■■■
■■■■■■■■■■■■■■■■■■

■

■ (REST API Calls)

▼

■■■
■■■■■■■■■■■■■■■■■■■■

■ WEB FRONTEND ■

■ - Displays invoices, creates manifests, generates reports ■

■ - Communicates with API server via REST endpoints ■

■■■
■■■■■■■■■■■■■■■■■■

■

▼

■■■
■■■■■■■■■■■■■■■■■■■■

■ OUTPUT GENERATION ■

■ - Excel dispatch reports (dispatch_report.js) ■

■ - PDF manifest documents ■

■ - Console/stored reports ■

■■■
■■■■■■■■■■■■■■■■■■■■

2.3 Database Schema Diagram

■■■
■■■■■■■■■■■■■■■■■■■■

■ ORDERS TABLE ■

■ ■■
■■■■■■■■■■■■■ ■

■ ■ id (PK) | filename | date_processed | customer_name | total_value ■ ■

■ ■ order_number | invoice_number | invoice_date | area | is_allocated ■ ■

■ ■ allocated_date | manifest_number | type | reference_number | ■ ■

■ ■ original_value | status | customer_number ■ ■

■ ■■
■■■■■■■■■■■■■ ■

■ ■ ■

■ ■ 1:N ■

■ ▼ ■

■ ■■
■■■■■■■■■■■■■ ■

■ ■ REPORT_ITEMS TABLE ■ ■

■ ■ ■■■
■■■■■■■■■■■ ■

■ ■ ■ id (PK) | report_id (FK) | invoice_number | order_number ■■ ■

■ ■ ■ customer_name | customer_number | invoice_date | area ■■ ■

■ ■ ■ sku | value | weight ■■ ■

■ ■ ■■■
■■■■■■■■■■■ ■

■ ■■
■■■■■■■■■■■■■ ■

■ ■ ■

■ ■ N:1 ■

■ ▼ ■

■ ■■
■■■■■■■■■■ ■

■ ■ REPORTS TABLE ■ ■

■ ■ ■■■
■■■■■■■■ ■

■ ■ ■ id (PK) | manifest_number | date | driver | assistant | checker ■■ ■

■ ■ ■ reg_number | pallets_brown | pallets_blue | crates | mileage ■■ ■

■ ■ ■ total_value | total_sku | total_weight | created_at ■■ ■

■ ■ ■■■
■■■■■■■■ ■

■ ■■
■■■■■■■■■■ ■

■ ■

■ ■■■■■■■■■■■ ■■■■■■■■■■■ ■■■■■■■■■■■
■■■■■■■■■■■■■■ ■

■ ■ USERS ■ ■ TRUCKS ■ ■SETTINGS ■ ■ MANIFEST_STAGING ■ ■

■ ■ TABLE ■ ■ TABLE ■ ■ TABLE ■ ■ TABLE ■ ■

■ ■■■■■■■■■■■ ■■■■■■■■■■■ ■■■■■■■■■■■
■■■■■■■■■■■■■■ ■

■ ■

■ ■■■■■■■■■■■
■■

■ ■CUSTOMER_ ■ ■ MANIFEST_EVENTS TABLE ■■

■ ■ROUTES TABLE■ ■ id (PK) | manifest_number | event_type | performed_by■■

■ ■■■■■■■■■■■
■■

■■
■■■■■■■■■■■■■■■

3. Component Descriptions

3.1 Backend Components

3.1.1 api_server.py (FastAPI Server)

Purpose: Main REST API server that handles all HTTP requests

Key Features:

- FastAPI-based REST API
- CORS middleware for cross-origin requests
- File watcher integration
- Request authentication via headers
- Comprehensive logging

Configuration:

python

MANIFEST_FOLDER = r"C:\Users\Assault\OneDrive\Documents\Delivery Route\Manifests_Output"

DEV_MODE = True

SERVER_START_TIME = datetime.now().isoformat()

Data Models:

- Invoice - Invoice data structure
- AllocateRequest - Allocation request payload
- LoginRequest - User authentication
- UserCreate / UserUpdate - User management
- TruckRequest - Vehicle information
- ReportRequest - Dispatch report data

3.1.2 database.py (SQLite Database Module)

Purpose: Database access layer for all CRUD operations

Key Functions:

3.1.3 file_watcher.py (Background Service)

Purpose: Monitor input folder for new PDF invoices

Configuration:

python

WATCH_FOLDER = r"\\BRD-DESKTOP-ELV\storage" # Network path

POLL_INTERVAL = 20 # seconds

FILE_STABILITY_CHECKS = 3

FILE_STABILITY_DELAY = 2 # seconds

Key Features:

- Polls folder every 20 seconds
- Verifies file stability (size consistency)
- Prevents reading partially written files
- Thread-safe operation
- Automatic reloading of invoice_processor

3.1.4 invoice_processor.py

Purpose: Process incoming PDF invoices and Excel manifests

Key Functions:

- Extract text from PDF invoices using PyPDF2
- Parse Excel manifest templates
- Extract invoice metadata (number, date, customer, value)
- Validate and normalize data
- Store in database
- Handle duplicate files
- Move processed files to processed folder

3.2 Frontend Components

3.2.1 index.html

Purpose: Main web interface for the delivery manifest system

Sections:

- Header: System branding and navigation
- Route Details: Manifest number, date, truck selection
- Invoice/Order Entry: Manual order input
- Available Invoices: List of pending invoices
- Current Manifest: Selected invoices for manifest
- Footer: System status and actions

Dependencies:

- Google Fonts (Inter)
- ExcelJS (for report generation)
- Lucide Icons

3.2.2 script.js

Purpose: Main application logic and API communication

Key Features:

- Invoice loading and filtering
- Manifest creation workflow
- Credit note handling
- Area-based filtering
- Order management
- Dispatch report generation

Modules:

- Invoice management
- Manifest staging
- Report generation
- Settings management

3.2.3 dispatch_report.js

Purpose: Generate Excel dispatch reports

Features:

- ExcelJS-based report generation
- Include driver, vehicle, cargo details
- Summarize invoices and totals
- Styling and formatting
- Downloadable output

3.2.4 style.css & dev_mode.css

Purpose: UI styling for the application

Features:

- Modern responsive design
- Card-based layout
- Form styling
- Button variants
- Status indicators

3.3 Utility Scripts

4. Database Schema

4.1 Table Definitions

4.1.1 orders Table

sql

```
CREATE TABLE orders (
id INTEGER PRIMARY KEY AUTOINCREMENT,
filename TEXT UNIQUE NOT NULL,
date_processed TEXT NOT NULL,
customer_name TEXT NOT NULL,
total_value TEXT DEFAULT '0.00',
order_number TEXT DEFAULT 'N/A',
invoice_number TEXT DEFAULT 'N/A',
invoice_date TEXT DEFAULT 'N/A',
area TEXT DEFAULT 'UNKNOWN',
is_allocated INTEGER DEFAULT 0,
allocated_date TEXT,
manifest_number TEXT,
type TEXT DEFAULT 'INVOICE',
reference_number TEXT,
original_value TEXT,
status TEXT DEFAULT 'PENDING',
customer_number TEXT DEFAULT 'N/A'
)
```

Fields Description:

4.1.2 reports Table

sql

```
CREATE TABLE reports (
id INTEGER PRIMARY KEY AUTOINCREMENT,
manifest_number TEXT UNIQUE NOT NULL,
date TEXT NOT NULL,
driver TEXT,
assistant TEXT,
```

checker TEXT,

reg_number TEXT,

pallets_brown INTEGER DEFAULT 0,

pallets_blue INTEGER DEFAULT 0,

crates INTEGER DEFAULT 0,

mileage INTEGER DEFAULT 0,

total_value REAL DEFAULT 0,

total_sku INTEGER DEFAULT 0,

total_weight REAL DEFAULT 0,

created_at TEXT NOT NULL

)

4.1.3 report_items Table

sql

CREATE TABLE report_items (

id INTEGER PRIMARY KEY AUTOINCREMENT,

report_id INTEGER NOT NULL,

invoice_number TEXT NOT NULL,

order_number TEXT,

customer_name TEXT,

customer_number TEXT,

invoice_date TEXT,

area TEXT,

sku INTEGER DEFAULT 0,

value REAL DEFAULT 0,

weight REAL DEFAULT 0,

FOREIGN KEY (report_id) REFERENCES reports(id)

)

4.1.4 users Table

sql

CREATE TABLE users (

id INTEGER PRIMARY KEY AUTOINCREMENT,

username TEXT UNIQUE NOT NULL,

password_hash TEXT NOT NULL,

is_admin INTEGER DEFAULT 0,

can_manifest INTEGER DEFAULT 1,

created_at TEXT NOT NULL

)

4.1.5 trucks Table

sql

CREATE TABLE trucks (

id INTEGER PRIMARY KEY AUTOINCREMENT,

reg TEXT UNIQUE NOT NULL,

driver TEXT,

assistant TEXT,

checker TEXT

)

4.1.6 settings Table

sql

CREATE TABLE settings (

id INTEGER PRIMARY KEY AUTOINCREMENT,

category TEXT NOT NULL,

value TEXT NOT NULL,

UNIQUE(category, value)

)

4.1.7 manifest_staging Table

sql

CREATE TABLE manifest_staging (

id INTEGER PRIMARY KEY AUTOINCREMENT,

session_id TEXT NOT NULL,

invoice_id INTEGER NOT NULL,

added_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,

FOREIGN KEY (invoice_id) REFERENCES orders(id)

)

Index:

sql

CREATE INDEX idx_staging_session ON manifest_staging(session_id)

4.1.8 manifest_events Table (Audit Trail)

sql

CREATE TABLE manifest_events (

id INTEGER PRIMARY KEY AUTOINCREMENT,

manifest_number TEXT NOT NULL,

event_type TEXT NOT NULL,

performed_by TEXT DEFAULT 'System',

timestamp TEXT NOT NULL

)

4.1.9 customer_routes Table

sql

CREATE TABLE customer_routes (

id INTEGER PRIMARY KEY AUTOINCREMENT,

customer_name TEXT UNIQUE NOT NULL,

route_name TEXT NOT NULL

)

4.2 Relationships

orders (1) ■■■■■■> (N) report_items (N) ■■■■■■> (1) reports

■

■ (via manifest_staging)

■

■■■■■■> manifest_events

users (1) ■■■■■■> (N) manifest_events

trucks (1) ■■■■■■> (N) reports

■

■■■■■■> settings (drivers, assistants, checkers)

5. API Endpoints

6. Data Flow

6.1 Invoice Processing Flow

1. File Watcher detects new PDF in Invoices_Input/

■■ Check file stability (3 consecutive size checks)

■■ Wait 2 seconds between checks

■■ Verify file is not locked

2. Trigger invoice_processor.main()

■■ Load PDF using PyPDF2

■■ Extract text from all pages

■■ Parse invoice data using regex patterns

■ ■■ Invoice number: #(\w+)

■ ■■ Order number: Order[:#\s](\w+)

■ ■■ Customer name patterns

■ ■■ Total value extraction

■ ■■ Date parsing

■■ Validate extracted data

■■ Store in database via database.add_order()

■■ Move file to Invoices_Processed/

3. Update frontend (optional real-time notification)

6.2 Manifest Creation Flow

1. User logs into web interface

■■ API authenticates via X-Username header

2. User loads available invoices

■■ GET /invoices (excludes staged invoices)

3. User filters by area/customer

■■ Client-side or API filtering

4. User selects invoices for manifest

■■ POST /invoices/allocate with filenames

■■ Added to manifest_staging table

■■ Invoices remain "available" in database

5. User confirms manifest creation

■■ POST /manifest/confirm

■■ Generate manifest_number (A{sequence})

■■ Update orders.manifest_number

■■ Set is_allocated = 1

■■ Clear staging entries

■■ Log event in manifest_events

6. User generates dispatch report

■■ POST /reports/generate

■■ Create report record

■■ Link report_items to invoices

■■ Calculate totals (SKU, weight, value)

■■ Return report data

6.3 Credit Note Handling Flow

1. User processes credit note PDF

■■ Same as invoice processing

■■ type = 'CREDIT_NOTE'

2. User can:

a) Full Credit - cancel original invoice

■■ POST /invoices/cancel

■■ Set status = 'CANCELLED'

b) Partial Credit - adjust value

■■ POST /invoices/update-value

■■ Update total_value

■■ Store original_value

■■ Update report totals

6.4 File Watcher Sequence Diagram

■■■■■■■■■■■■■■■ ■■■■■■■■■■■■■■■ ■■■■■■■■■■■■■■■■■ ■■■■■■■■■■■■■■■■■■■

■ Folder ■ ■File Watcher ■ ■Invoice Proc. ■ ■ Database ■

■■■■■■■■■■■■■■■ ■■■■■■■■■■■■■■■ ■■■■■■■■■■■■■■■■■ ■■■■■■■■■■■■■■■■■■■

■ ■ ■ ■

■ Poll every 20s ■ ■ ■

■■■■■■■■■■■■■■■■■■■■■■>■ ■ ■

■ ■ ■ ■

■ List files ■ ■ ■

■<■■■■■■■■■■■■■■■■■■■■ ■ ■

■ ■ ■ ■

■ New file detected ■ ■ ■

■■■■■■■■■■■■■■■■■■■■■■>■ ■ ■

■ ■ ■ ■

■ Check stability ■ ■ ■

■ (3 checks, 2s ea) ■ ■ ■

■ ■ ■ ■

■ File stable ■ ■ ■

■■■■■■■■■■■■■■■■■■■■■■>■ ■ ■

■ ■ ■ ■

■ Trigger processing ■■■■■■■■■■■■■■■■■■■■■■>■ ■

■ ■ ■ ■

■ ■ Extract PDF data ■ ■

■ ■ ■ ■

■ ■ Parse text ■ ■

■ ■ ■ ■

■ ■ Validate ■ ■

■ ■ ■ ■

■ ■ Store in DB ■■■■■■■■■■■■■■■■■■■■■■>■

■ ■ ■ ■

■ ■ Move to processed ■ ■

■ ■ ■ ■

■ ■ Return success ■<■■■■■■■■■■■■■■■■■■■■■

■ ■<■■■■■■■■■■■■■■■■■■■■■■ ■

■ ■ ■ ■

7. Key Features

7.1 Invoice Management

- Automatic Detection: File watcher automatically detects new PDF invoices
- Text Extraction: PyPDF2-based text extraction from invoices
- Regex Parsing: Pattern-based data extraction for invoice fields
- Area Classification: Automatic area assignment based on customer
- Multi-type Support: Handles both invoices and credit notes
- Duplicate Prevention: Unique filename constraint prevents duplicates
- Search Functionality: Search by invoice number, order number, or customer

7.2 Manifest Workflow

- Staging Area: Invoices remain "available" until manifest is confirmed
- Session-based Staging: Multiple users can build manifests independently
- Auto-numbering: Automatic manifest number generation (e.g., A35426)
- Flexible Selection: Filter by area before adding to manifest
- Undo Capability: Remove invoices from staging before confirming
- Allocation Tracking: Full history of invoice allocations

7.3 Credit Note Support

- Full Credit: Mark entire invoice as cancelled
- Partial Credit: Adjust invoice value while keeping active
- Original Value Tracking: Store original value for audit trail
- Status Management: Track cancelled vs. pending vs. allocated

7.4 Dispatch Reporting

- Excel Export: Generate formatted Excel dispatch reports
- Driver/Vehicle Info: Include driver, assistant, checker names
- Cargo Details: Pallets (brown/blue), crates, mileage
- Invoice Summary: SKU count, total weight, total value
- Customer Details: Customer numbers and delivery areas

7.5 User Management

- Role-based Access: Admin and regular user roles
- Permission Control: Manifest creation permissions
- Secure Storage: Password hashing using SHA-256
- Activity Logging: Track user actions via manifest events

7.6 System Features

- Auto-start File Watcher: File watcher starts with API server
- Real-time Updates: Polling-based frontend updates
- Responsive Design: Modern UI with mobile support
- Error Handling: Comprehensive error logging
- Data Validation: Input validation on both client and server

8. Technical Specifications

8.1 File Structure

Delivery Route/

■■■ api_server.py # FastAPI REST server

■■■ database.py # SQLite database module

■■■ invoice_processor.py # PDF/Excel processing

■■■ file_watcher.py # Background file watcher

■■■ index.html # Main web interface

■■■ script.js # Frontend logic

■■■ style.css # UI styles

■■■ dev_mode.css # Development styles

■■■ dispatch_report.js # Excel report generation

■■■ footer.js # Footer component

■■■ delivery.db # SQLite database

■■■ requirements.txt # Python dependencies

■

■■■ Invoices_Input/ # New invoices (auto-monitored)

■ ■■■ .pdf, .xlsx

■

■■■ Invoices_Processed/ # Processed invoices

■ ■■■ .pdf

■

■■■ Manifests_Output/ # Generated manifests

■ ■■■ Manifest_.xlsx

■

■■■ debug_.py # Debugging scripts

■■■ test_.py # Test scripts

■■■ migrate_.py # Migration scripts

■

■■■ Documentation/

■■■ .md # Various documentation files

8.2 Dependencies (requirements.txt)

fastapi>=0.100.0

uvicorn>=0.22.0

python-multipart>=0.0.6

pydantic>=2.0.0

PyPDF2>=3.0.0

openpyxl>=3.1.0

pandas>=2.0.0

8.3 API Server Configuration

python

FastAPI Server

HOST = "0.0.0.0"

PORT = 8000

DEBUG = True

CORS Configuration

allow_origins = [""] # Configure for production

allow_methods = [""]

allow_headers = [""]

File Watcher

POLL_INTERVAL = 20 # seconds

WATCH_FOLDER = r"\\BRD-DESKTOP-ELV\storage"

8.4 Database Configuration

python

DB_PATH = "delivery.db" # Relative to project root

Full path: C:\Users\Assault\OneDrive\Documents\Delivery Route\delivery.db

8.5 Performance Considerations

8.6 Security Considerations

- Authentication: X-Username header-based auth
- Password Storage: SHA-256 hashed passwords
- CORS: Configured for development (needs restriction for production)
- Input Validation: Pydantic models validate all requests
- SQL Injection Prevention: Parameterized queries (sqlite3)
- File Validation: File stability checks before processing

8.7 Backup & Recovery

- Database: SQLite file can be backed up manually
- Settings: Stored in database
- Processed Files: Archived in Invoices_Processed/
- Logs: server.log and file_watcher.log

Appendix A: Regex Patterns

The system uses the following regex patterns for invoice parsing:

python

Invoice number patterns

invoice_patterns = [
r'#(\w+)',
r'Invoice\s[:#]?\s(\w+)',
r'INV[:#]?\s(\w+)'
]

Order number patterns

order_patterns = [
r'Order[:#\s](\w+)',
r'Order\s+Number[:#]?\s(\w+)'
]

Customer name patterns

customer_patterns = [

r'(?:Sold to|Bill to|Customer)[^a-zA-Z]([A-Z][A-Z\s\.]+(?:PT\s?LTD|PVT\s?LTD|CORP|LTD|CC)?)',

r'^([A-Z][A-Z\s\.]+(?:PT\s?LTD|PVT\s?LTD|CORP|LTD|CC)?)'

]

Total value patterns

value_patterns = [

r'Total[:\s]\$?([\d,]+\.?\d)',

r'Grand\s+Total[:\s]\$?([\d,]+\.?\d)',

r'Amount\s+Due[:\s]\$?([\d,]+\.?\d)'

]

Appendix B: Glossary

Appendix C: Version History

Document Generated: February 8, 2026

Generated By: System Analysis

Classification: Internal Use Only

End of Document