Unified Warehouse Management System (WMS)

Project Duration: 4 Months | **Completion Date:** May 2025

1. Executive Summary

The Unified Warehouse Management System (WMS) is a comprehensive multi-component solution designed to streamline warehouse operations, enhance dispatch management between branches, and provide robust employee and device monitoring capabilities. Developed over a four-month period, this system integrates an Android mobile application for warehouse floor operations, backend APIs for centralized data management, and two specialized web applications for dispatch and employee management services.

The WMS addresses critical operational challenges by providing a cohesive platform that ensures real-time inventory tracking, efficient order processing, and transparent inter-branch logistics coordination. By leveraging modern technology stacks and a distributed architecture, the system delivers a scalable, reliable solution capable of supporting growing operational demands while maintaining data integrity across all components.

2. System Architecture

The WMS employs a distributed, multi-platform architecture designed for flexibility and clear separation of concerns. The system consists of six primary components that interact seamlessly to deliver a cohesive user experience across different operational domains.

2.1. Architecture Overview

System Architecture Diagram Description: The WMS architecture consists of three main tiers interconnected through a central database:

- Mobile Tier: .NET MAUI Android application connecting to backend services via REST API
- API Tier: ASP.NET Core Web API as the central data management service
- Web Application Tier: Two Django applications (Dispatch Management and Employee Management)
- Shared Components: A .NET Standard library providing common models and business logic
- Database Tier: PostgreSQL database hosted on Digital Ocean serving all components

The architecture is structured as follows:

- **Mobile Tier (`wms_android` & SDK Bindings):** A .NET MAUI application optimized for Android warehouse devices (CS30 and A90 POS terminals) with hardware integration capabilities.
- **API Tier (`wms_android.api`):** An ASP.NET Core Web API serving as the central backend service handling data operations and business logic.

- **Web Tier:**
  - `wms_web.dispatch`: A Django-based web application for branch managers to oversee dispatch operations.
  - `wms_web.ems`: A Django-based Employee Management System for user administration and activity monitoring.
- **Shared Logic Tier (`wms_android.shared`):** A .NET 8.0 standard library containing common data models and business logic.
- **Database Tier:** A centralized relational database hosted on Digital Ocean.

2.2. Data Flow

Data Flow Diagram Description: The system's data flows through several key pathways:

1. **Warehouse Operations Flow:**
   - Warehouse staff use the Android MAUI app installed on CS30/A90 devices
   - App communicates with backend API for data operations (inventory updates, order processing)
   - Native device features (scanners, printers) accessed via SDK binding libraries
   - API persists all operations to central PostgreSQL database
2. **Dispatch Management Flow:**
   - Branch managers access web-based dispatch portal
   - Dispatch application reads/writes directly to central database
   - Real-time updates visible to both web users and mobile app users
   - Notification system alerts relevant parties of status changes
3. **Employee Management Flow:**
   - Administrators register users and devices through EMS web portal
   - System tracks device usage by serial number
   - Performance metrics collected from user activities
   - Reports generated for management review

The system's data flows through several pathways:

1. **Warehouse Operations Flow:** Warehouse staff use the Android MAUI app to scan inventory, process orders, and manage stock movements. The app communicates with the ASP.NET Core API, which persists data to the central database.
2. **Dispatch Management Flow:** Branch managers utilize the Django dispatch web application to create, track, and manage inter-branch transfers. This application interacts with the central database to maintain dispatch records.

3. **Employee Management Flow:** Administrators use the Django EMS web application to register users, monitor device usage, and analyze user activity. This component maintains records of user logins and performance metrics.

3. Technologies Used

The WMS leverages a diverse technology stack to address the varied requirements of each component:

3.1. Mobile Application (`wms_android`)

- **Framework:** .NET MAUI (targeting Android)
- **Language:** C# 12
- **Architecture Pattern:** Model-View-ViewModel (MVVM)
- **Key Libraries:**
  - CommunityToolkit.Mvvm for MVVM implementation
  - SQLite for local caching
  - RestSharp for API communication

3.2. Device SDK Integration (`AndroidPosSdk`, `VanstonePosSdk`)

- **Technology:** .NET for Android bindings
- **Original SDKs:** Java-based SDKs for CS30 and A90 devices
- **Implementation:** Custom binding libraries that expose native device functionality to the .NET MAUI application

3.3. Backend API (`wms_android.api`)

- **Framework:** ASP.NET Core 8.0 Web API
- **Language:** C# 12
- **Database ORM:** Entity Framework Core 8.0
- **Authentication:** JWT-based token authentication
- **API Design:** RESTful principles with versioning

3.4. Shared Library (`wms_android.shared`)

- **Framework:** .NET 8.0 Standard
- **Language:** C# 12
- **Components:** Data models, DTOs, and shared business logic

3.5. Web Applications (`wms_web.dispatch`, `wms_web.ems`)

- **Framework:** Django 4.2

- **Language:** Python 3.11
- **Frontend:** HTML5, CSS3, JavaScript, Bootstrap 5
- **Authentication:** Django's built-in authentication system

3.6. Database

- **System:** PostgreSQL 16
- **Hosting:** Digital Ocean Managed Database
- **Migration Management:**
  - Entity Framework Core migrations for .NET components
  - Django migrations for Python components

3.7. Development & Deployment

- **Version Control:** Git with GitHub
- **CI/CD:** GitHub Actions
- **Development Environments:**
  - Visual Studio 2022 for .NET components
  - VS Code with Python extensions for Django components
- **Deployment:** Docker containers on Digital Ocean Kubernetes

4. Component Deep Dive

4.1. Mobile Application (`wms_android`)

The .NET MAUI Android application serves as the primary interface for warehouse staff, providing a comprehensive set of tools for daily operations on dedicated POS devices.

4.1.1. Key Features

- **Authentication:** Secure user login with role-based access control
- **Inventory Management:**
  - Real-time stock level visibility
  - Barcode scanning for rapid item identification
  - Cycle counting with discrepancy resolution
  - Stock adjustments with reason codes
- **Order Processing:**
  - Pick list generation and management
  - Order picking with scan verification
  - Packing station integration

  - Shipping label generation
- **Receiving:**
  - Purchase order validation
  - Item verification via barcode scanning
  - Discrepancy reporting
  - Receipt generation
- **Dispatch Management:**
  - Creation of inter-branch transfers
  - Dispatch confirmation and receipt
  - Transfer status tracking

4.1.2. Offline Capability

The application implements a local SQLite database for offline operation, synchronizing with the central system when connectivity is restored.

4.1.3. UI/UX Design

The interface is optimized for industrial POS devices with:

- Large touch targets for gloved operation
- High-contrast color scheme for warehouse visibility
- Minimized text input requirements
- Context-sensitive help

4.2. Device SDK Integration

The system incorporates two custom binding libraries that enable the .NET MAUI application to access hardware features of specialized warehouse devices.

4.2.1. AndroidPosSdk (CS30 Integration)

- Enables access to the CS30 device's:
  - 1D/2D barcode scanner
  - Thermal printer
  - RFID reader
  - NFC capabilities

4.2.2. VanstonePosSdk (A90 Integration)

- Provides interfaces to the A90 device's:
  - High-speed barcode scanner

- Receipt printer
- Signature capture pad
- Payment terminal integration

4.2.3. Implementation Approach

The binding libraries are generated from the manufacturers' Java SDKs using .NET for Android binding techniques. This approach maintains native performance while allowing seamless integration with the .NET MAUI codebase.

4.3. Backend API (`wms_android.api`)

The ASP.NET Core Web API serves as the central nervous system for data management and business logic execution.

4.3.1. API Endpoints

The API provides comprehensive RESTful endpoints organized by domain:

- `/api/v1/auth` - Authentication and authorization
- `/api/v1/inventory` - Inventory management operations
- `/api/v1/orders` - Order processing and fulfillment
- `/api/v1/receiving` - Goods receipt management
- `/api/v1/dispatch` - Inter-branch transfer operations
- `/api/v1/users` - User management
- `/api/v1/devices` - Device registration and tracking

4.3.2. Security Features

- JWT token-based authentication
- Role-based authorization
- Request validation and sanitization
- API rate limiting
- Comprehensive audit logging

4.3.3. Performance Optimization

- Response caching for frequently accessed data
- Asynchronous request handling
- Pagination for large data sets
- Compression for bandwidth optimization

4.4. Shared Library (`wms_android.shared`)

This .NET 8.0 standard library ensures consistency across the mobile application and API by providing:

4.4.1. Data Models

- Core business entities reflecting database schema
- Data Transfer Objects (DTOs) for API communication
- Validation attributes for data integrity

4.4.2. Shared Logic

- Common calculation algorithms
- Business rule implementations
- Extension methods
- Constants and enumeration definitions

4.5. Dispatch Web Application (`wms_web.dispatch`)

The Django-based dispatch management portal enables branch managers to coordinate inter-branch transfers efficiently.

4.5.1. Key Features

- **Dashboard:** Real-time overview of incoming and outgoing dispatches
- **Dispatch Creation:** Intuitive workflow for creating new transfers
- **Inventory Visibility:** Real-time stock levels across branches
- **Status Tracking:** Timeline view of dispatch progress
- **Reporting:** Comprehensive dispatch performance metrics
- **Notifications:** Email and in-app alerts for status changes

4.5.2. Integration Points

- Communicates with the central database for dispatch data
- Potentially interacts with the ASP.NET Core API for certain operations

4.6. Employee Management System (`wms_web.ems`)

This Django application provides administrative capabilities for user management and activity monitoring.

4.6.1. Key Features

- **User Administration:**

    - User registration and profile management
    - Role and permission assignment
    - Password policy enforcement
- **Device Tracking:**
    - Device-to-user association
    - Login tracking by device serial number
    - Usage patterns analysis
- **Performance Monitoring:**
    - User activity metrics
    - Productivity analysis
    - Operational efficiency reports
- **Audit Trail:**
    - Comprehensive activity logging
    - Security event tracking
    - Compliance reporting

5. Database Design

The system utilizes a centralized PostgreSQL database hosted on Digital Ocean, serving as the single source of truth for all components.

5.1. Schema Overview

Entity Relationship Diagram Description: The database schema encompasses several interconnected domains with the following key entities and relationships:

1. **User Management Domain:**
    - Users (PK: UserID, FK: RoleID)
    - Roles (PK: RoleID)
    - Permissions (PK: PermissionID)
    - UserPermissions (PK: UserID, PermissionID)
2. **Inventory Domain:**
    - Products (PK: ProductID, FK: CategoryID)
    - Categories (PK: CategoryID)
    - Stock (PK: StockID, FK: ProductID, LocationID)
    - Locations (PK: LocationID, FK: BranchID)
    - Branches (PK: BranchID)

3. **Orders Domain:**
    - Orders (PK: OrderID, FK: UserID, BranchID)
    - OrderItems (PK: OrderItemID, FK: OrderID, ProductID)
    - OrderStatus (PK: StatusID)
    - OrderStatusHistory (PK: HistoryID, FK: OrderID, StatusID, UserID)
4. **Receiving Domain:**
    - PurchaseOrders (PK: POID, FK: SupplierID, BranchID)
    - Receipts (PK: ReceiptID, FK: POID, UserID)
    - ReceiptItems (PK: ReceiptItemID, FK: ReceiptID, ProductID)
5. **Dispatch Domain:**
    - Transfers (PK: TransferID, FK: SourceBranchID, DestinationBranchID, CreatedByUserID)
    - TransferItems (PK: TransferItemID, FK: TransferID, ProductID)
    - TransferStatus (PK: StatusID)
    - TransferStatusHistory (PK: HistoryID, FK: TransferID, StatusID, UserID)
6. **Device Management Domain:**
    - Devices (PK: DeviceID, Fields: SerialNumber, DeviceType)
    - DeviceLogs (PK: LogID, FK: DeviceID, UserID)
7. **Activity Tracking Domain:**
    - UserSessions (PK: SessionID, FK: UserID, DeviceID)
    - ActivityLogs (PK: LogID, FK: UserID, ActivityTypeID)
    - ActivityTypes (PK: ActivityTypeID)

The database schema encompasses several key domains:

- **User Management:** Users, Roles, Permissions
- **Inventory:** Products, Categories, Stock, Locations
- **Orders:** Orders, Order Items, Order Status
- **Receiving:** Purchase Orders, Receipts, Receipt Items
- **Dispatch:** Transfers, Transfer Items, Transfer Status
- **Device Management:** Devices, Device Logs
- **Activity Tracking:** User Sessions, Activity Logs

5.2. Data Management Strategy

- **Schema Evolution:**
    - Entity Framework Core migrations manage schema changes for .NET components

    - Django migrations handle schema changes for Python components
- **Data Integrity:**
    - Foreign key constraints ensure relational integrity
    - Check constraints enforce business rules at the database level
- **Performance Optimization:**
    - Strategic indexing based on query patterns
    - Partitioning for large tables
    - Database views for complex reporting queries

6. Security Considerations

6.1. Authentication & Authorization

- Multi-factor authentication for administrative access
- Role-based access control across all components
- Least privilege principle enforcement

6.2. Data Protection

- Encryption of sensitive data at rest
- TLS encryption for all data in transit
- Secure API communication with token validation

6.3. Audit & Compliance

- Comprehensive audit logging across all components
- User activity tracking with device association
- Regular security review procedures

7. Development Methodology

The project was executed using an Agile methodology with two-week sprints, enabling rapid development while maintaining quality:

7.1. Project Timeline

- **Month 1:** Requirements gathering, architecture design, database schema development
- **Month 2:** Core API development, mobile application foundation, SDK integration
- **Month 3:** Web application development, integration testing, performance optimization
- **Month 4:** User acceptance testing, bug fixing, deployment preparation, documentation

7.2. Team Composition

- 2 .NET Developers (Mobile & API)
- 2 Python/Django Developers (Web Applications)
- 1 Database Administrator
- 1 DevOps Engineer
- 1 Project Manager

8. Future Enhancements

Several potential enhancements have been identified for future iterations:

- **Advanced Analytics:** Implementation of business intelligence dashboards
- **Machine Learning Integration:** Demand forecasting and inventory optimization
- **Mobile Expansion:** iOS support for BYOD environments
- **IoT Integration:** Warehouse sensor network for environmental monitoring
- **Voice Picking:** Voice-directed warehouse operations
- **AR Integration:** Augmented reality for complex picking operations

9. Conclusion

The Unified Warehouse Management System represents a significant advancement in warehouse operations technology. By seamlessly integrating mobile applications, backend services, and web portals, the system delivers a comprehensive solution that addresses the unique challenges of modern warehouse management.

The successful completion of this complex project within a four-month timeframe demonstrates the effectiveness of the chosen architecture, technologies, and development methodology. The system's modular design ensures scalability and adaptability as business requirements evolve, providing a solid foundation for future enhancements.

Appendices

Appendix A: Technical Documentation References

- CS30 SDK Documentation: `CS30Pro-SDK instructions V1.0.1.pdf`
- A90 SDK Documentation: `《Vanstone Android POS API programming manual v2.00》.docx.pdf`
- API Documentation: [Internal Wiki Link]
- Database Schema Documentation: [Internal Wiki Link]

Appendix B: Deployment Architecture

- Containerization strategy
- Kubernetes configuration
- CI/CD pipeline details
- Monitoring and alerting setup

Appendix C: Testing Strategy

- Unit testing coverage
- Integration testing approach
- Load testing results
- User acceptance testing outcomes