Logistics Warehouse Management System (LWMS) - Project Documentation

Table of Contents

Project Overview

LWMS (Logistics Warehouse Management System) is a comprehensive Spring Boot-based web application designed to manage warehouse operations efficiently. The system provides end-to-end solutions for inventory management, shipment tracking, space optimization, maintenance scheduling, and reporting.

Key Features

- **Inventory Management**: Track items, quantities, locations, and categories
- **Shipment Handling**: Manage incoming and outgoing shipments with tracking
- **Space Optimization**: Monitor warehouse capacity and space allocation
- **Maintenance Scheduling**: Schedule and track equipment maintenance
- **Reporting System**: Generate comprehensive reports for business insights
- **User Authentication**: Secure access control with role-based permissions
- **Responsive UI**: Modern, responsive web interface with dark/light themes

Business Value

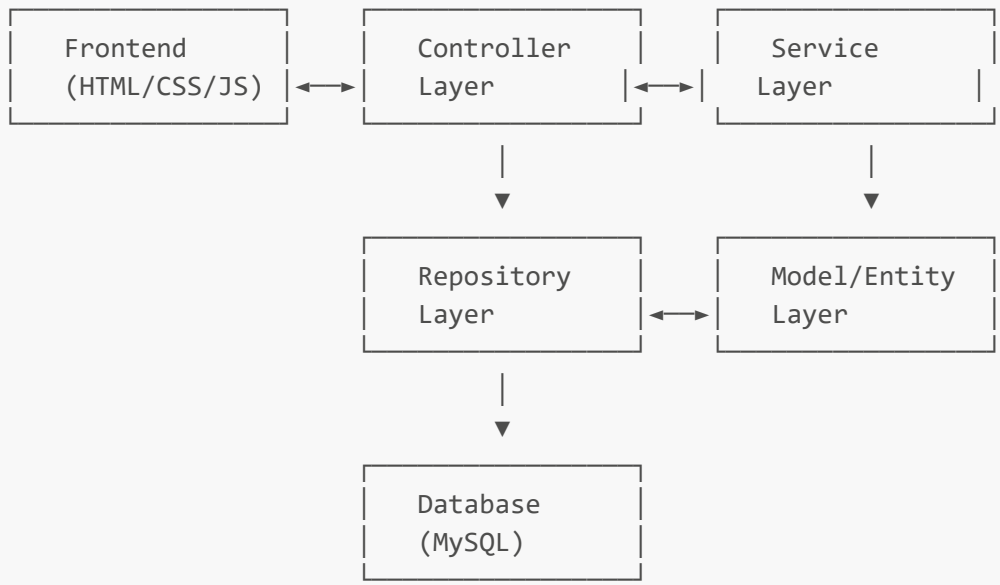
- Streamline warehouse operations
- Improve inventory accuracy
- Enhance shipment tracking
- Optimize space utilization
- Reduce operational costs
- Provide real-time insights

System Architecture

Architecture Pattern

- **Layered Architecture** with clear separation of concerns
- **MVC Pattern** for web layer
- **Repository Pattern** for data access
- **Service Layer** for business logic

Components



Technology Stack

Backend

- **Java 21**: Latest LTS version for robust enterprise development
- **Spring Boot 3.5.3**: Rapid application development framework
- **Spring Data JPA**: Data access layer with Hibernate
- **Spring Web**: RESTful web services
- **Spring Validation**: Input validation and data integrity
- **SpringDoc OpenAPI**: API documentation and testing

Frontend

- **HTML5**: Semantic markup
- **CSS3**: Modern styling with custom themes
- **JavaScript (ES6+)**: Dynamic functionality
- **Bootstrap 5.3.3**: Responsive UI framework
- **Chart.js**: Data visualization
- **Font Awesome**: Icon library

Database

- **MySQL 8.0**: Relational database management system
- **Hibernate**: Object-relational mapping (ORM)

Build Tool

- **Maven**: Dependency management and build automation

Development Tools

- **Spring Boot DevTools**: Hot reload and development utilities
- **Spring Boot Test**: Testing framework

Project Structure

```
lwms/
├── src/
│   ├── main/
│   │   ├── java/com/cts/lwms/
│   │   │   ├── controller/          # REST API Controllers
│   │   │   ├── service/             # Business Logic Layer
│   │   │   ├── repo/                # Data Access Layer
│   │   │   ├── model/               # Entity Classes
│   │   │   ├── dto/                 # Data Transfer Objects
│   │   │   ├── LwmsApplication.java # Main Application Class
│   │   │   └── WebConfig.java       # Web Configuration
│   │   └── resources/
│   │       ├── static/              # Static Web Resources
│   │       │   ├── admin/           # Admin Interface
│   │       │   ├── css/             # Stylesheets
│   │       │   ├── js/              # JavaScript Files
│   │       │   └── assets/          # Images and Media
│   │       ├── templates/           # Thymeleaf Templates
│   │       └── application.properties # Configuration
│   └── test/                        # Test Cases
├── pom.xml                          # Maven Configuration
├── mvnw                            # Maven Wrapper (Unix)
└── mvnw.cmd                        # Maven Wrapper (Windows)
```

Database Design

Entity Relationships

1. Inventory Entity

```
CREATE TABLE Inventory (
    itemId INT PRIMARY KEY AUTO_INCREMENT,
    itemName VARCHAR(255) NOT NULL,
    category VARCHAR(255),
    quantity INT,
    location VARCHAR(255),
    lastUpdated TIMESTAMP,
```

```
    UNIQUE KEY uk_item_name (itemName)
);
```

Fields:

- `itemId`: Primary key, auto-generated
- `itemName`: Name of the inventory item
- `category`: Item category classification
- `quantity`: Available quantity in stock
- `location`: Storage location in warehouse
- `lastUpdated`: Timestamp of last modification

2. Shipment Entity

```
CREATE TABLE Shipment (
    shipmentId INT PRIMARY KEY AUTO_INCREMENT,
    origin VARCHAR(255),
    destination VARCHAR(255),
    status VARCHAR(50),
    expectedDeliveryDate TIMESTAMP,
    itemId INT NOT NULL,
    FOREIGN KEY (itemId) REFERENCES Inventory(itemId)
);
```

Fields:

- `shipmentId`: Primary key, auto-generated
- `origin`: Shipment origin location
- `destination`: Shipment destination
- `status`: Current shipment status
- `expectedDeliveryDate`: Expected delivery date
- `itemId`: Foreign key to Inventory

3. Space Entity

```
CREATE TABLE Space (
    spaceId INT PRIMARY KEY AUTO_INCREMENT,
    totalCapacity INT,
    usedCapacity INT,
    availableCapacity INT,
    zone VARCHAR(255)
);
```

Fields:

- `spaceId`: Primary key, auto-generated

- `totalCapacity`: Total warehouse capacity
- `usedCapacity`: Currently used capacity
- `availableCapacity`: Available capacity
- `zone`: Warehouse zone identifier

4. MaintenanceSchedule Entity

```
CREATE TABLE Maintenance (
    scheduleId INT PRIMARY KEY AUTO_INCREMENT,
    equipmentId INT,
    taskDescription TEXT,
    scheduledDate TIMESTAMP,
    completionStatus VARCHAR(50)
);
```

Fields:

- `scheduleId`: Primary key, auto-generated
- `equipmentId`: Equipment identifier
- `taskDescription`: Maintenance task details
- `scheduledDate`: Scheduled maintenance date
- `completionStatus`: Current completion status

5. Report Entity

```
CREATE TABLE Report (
    reportId INT PRIMARY KEY AUTO_INCREMENT,
    reportType VARCHAR(255),
    generatedOn TIMESTAMP,
    details LONGTEXT
);
```

Fields:

- `reportId`: Primary key, auto-generated
- `reportType`: Type of report generated
- `generatedOn`: Report generation timestamp
- `details`: Report content (LONGTEXT for large content)

Database Configuration

```
spring.datasource.url=jdbc:mysql://localhost:3306/lwms?
useSSL=false&serverTimezone=UTC
spring.datasource.username=root
spring.datasource.password=root
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
```

```
spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL8Dialect
```

API Documentation

Base URL

```
http://localhost:8080
```

1. Home Controller

Purpose: Handles main application routing and authentication

Endpoints:

- `GET /` → Redirects to `/admin/dashboard.html` (Landing page)
- `GET /admin` → Redirects to `/admin/home.html` (User dashboard after login)

2. Inventory Controller

Purpose: Manages warehouse inventory operations

Base Path: `/inventory`

Endpoints:

Method	Endpoint	Description	Request Body	Response
POST	`/add`	Add new inventory item	Inventory object	Created Inventory
PUT	`/update`	Update existing item	InventoryDTO	Updated Inventory
DELETE	`/remove/{itemId}`	Remove item by ID	-	No Content (204)
GET	`/view`	View all inventory	-	List of Inventory

Request/Response Examples:

Add Item:

```
POST /inventory/add
{
    "itemName": "Laptop",
    "category": "Electronics",
    "quantity": 50,
    "location": "A1-B2"
}
```

Update Item:

```
PUT /inventory/update
{
    "itemId": 1,
    "itemName": "Laptop Pro",
    "category": "Electronics",
    "quantity": 45,
    "location": "A1-B2"
}
```

3. Shipment Controller

Purpose: Manages shipment operations and tracking

Base Path: `/shipment`

Endpoints:

Method	Endpoint	Description	Request Body	Response
POST	`/receive`	Receive incoming shipment	ShipmentDTO	Created Shipment
PUT	`/dispatch`	Dispatch outgoing shipment	ShipmentDTO	Updated Shipment
GET	`/track/{shipmentId}`	Track shipment by ID	-	Shipment details
GET	`/all`	Get all shipments	-	List of Shipments

Request/Response Examples:

Receive Shipment:

```
POST /shipment/receive
{
    "itemId": 1,
    "origin": "Supplier A",
    "destination": "Warehouse",
    "status": "Received",
    "expectedDeliveryDate": "2024-01-15T10:00:00"
}
```

Dispatch Shipment:

```
PUT /shipment/dispatch
{
    "itemId": 1,
    "origin": "Warehouse",
    "destination": "Customer B",
    "status": "Dispatched",
    "expectedDeliveryDate": "2024-01-20T14:00:00"
}
```

4. Space Controller

Purpose: Manages warehouse space allocation and optimization

Base Path: `/space`

Endpoints:

Method	Endpoint	Description	Request Body	Response
GET	`/view`	View space usage	-	List of Space
POST	`/allocate`	Allocate new space	Space object	Created Space
DELETE	`/free/{spaceId}`	Free allocated space	-	No Content (204)

Request/Response Examples:

Allocate Space:

```
POST /space/allocate
{
    "totalCapacity": 1000,
    "usedCapacity": 0,
    "availableCapacity": 1000,
    "zone": "Zone A"
}
```

5. Maintenance Controller

Purpose: Manages equipment maintenance scheduling

Base Path: `/maintenance`

Endpoints:

Method	Endpoint	Description	Request Body	Response

Method	Endpoint	Description	Request Body	Response
POST	`/schedule`	Schedule maintenance	MaintenanceSchedule	Created Schedule
PUT	`/update`	Update maintenance	MaintenanceSchedule	Updated Schedule
GET	`/view`	View all schedules	-	List of Schedules
GET	`/get/{scheduleId}`	Get schedule by ID	-	MaintenanceSchedule

Request/Response Examples:

Schedule Maintenance:

```
POST /maintenance/schedule
{
    "equipmentId": 101,
    "taskDescription": "Monthly inspection of forklift",
    "scheduledDate": "2024-01-15T09:00:00",
    "completionStatus": "Scheduled"
}
```

6. Report Controller

Purpose: Generates and manages system reports

Base Path: `/report`

Endpoints:

Method	Endpoint	Description	Request Body	Response
POST	`/generate`	Generate new report	Report object	Created Report
GET	`/get/{reportId}`	Get report by ID	-	Report details
GET	`/all`	Get all reports	-	List of Reports

Request/Response Examples:

Generate Report:

```
POST /report/generate
{
    "reportType": "Inventory Summary",
    "details": "Monthly inventory summary report"
}
```

Frontend Documentation

1. Dashboard (Landing Page)

File: `/admin/dashboard.html` **Purpose**: Public landing page with login form

Features:

- Responsive design with Bootstrap 5
- Dark/light theme toggle
- Login form for user authentication
- Animated SVG graphics
- Modern UI with shadow effects

Key Elements:

- Navigation bar with logo
- Login form with username/password fields
- Theme toggle button
- Animated truck illustration

2. Home (User Dashboard)

File: `/admin/home.html` **Purpose**: Main application interface after login

Features:

- Sidebar navigation with collapsible menu
- Tab-based content management
- Real-time data display
- Search functionality
- Theme switching (light/dark mode)
- Toast notifications
- Loading spinners

Navigation Sections:

1. **Dashboard**: Overview and key metrics
2. **Inventory Management**: Item management interface
3. **Space Optimization**: Warehouse space monitoring
4. **Shipment Handling**: Shipment operations
5. **Maintenance Scheduling**: Equipment maintenance
6. **Reports**: Report generation and viewing

UI Components:

- Responsive sidebar navigation
- Top navigation bar with search
- Content sections with tab switching
- Toast notification system
- Loading indicators

- Theme toggle switch

3. Styling

CSS Files:

- `/css/style.css`: Global styles and common components
- `/css/admin.css`: Admin-specific styling

Features:

- CSS custom properties for theming
- Responsive grid layouts
- Smooth transitions and animations
- Dark/light mode support
- Mobile-first responsive design

4. JavaScript Functionality

JS Files:

- `/js/script.js`: Common functionality
- `/js/admin.js`: Admin-specific features

Features:

- Tab navigation system
- Theme switching
- Form handling
- API integration
- Dynamic content loading
- Search functionality
- Toast notifications

Configuration

Application Properties

```
# Application Name
spring.application.name=lwms

# Server Configuration
server.port=8080

# Static Resource Configuration
spring.web.resources.static-locations=classpath:/static/
spring.mvc.static-path-pattern=/**

# Database Configuration
spring.datasource.url=jdbc:mysql://localhost:3306/lwms?
useSSL=false&serverTimezone=UTC
```

```
spring.datasource.username=root
spring.datasource.password=root
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver

# JPA/Hibernate Configuration
spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL8Dialect
```

Maven Configuration

Key Dependencies:

- Spring Boot Starter Web
- Spring Boot Starter Data JPA
- Spring Boot Starter Thymeleaf
- Spring Boot Starter Validation
- Spring Boot DevTools
- MySQL Connector
- SpringDoc OpenAPI

Setup and Installation

Prerequisites

1. **Java 21** or higher
2. **Maven 3.6+** or use Maven wrapper
3. **MySQL 8.0+** database server
4. **Git** for version control

Installation Steps

1. Clone the Repository

```
git clone <repository-url>
cd lwmsfinal-code/lwms
```

2. Database Setup

```
-- Create database
CREATE DATABASE lwms;
USE lwms;

-- Tables will be auto-created by Hibernate
-- Or run the SQL scripts manually if needed
```

3. Configure Database

Edit `src/main/resources/application.properties`:

```
spring.datasource.username=your_username
spring.datasource.password=your_password
```

4. Build and Run

```
# Using Maven wrapper
./mvnw spring-boot:run

# Or using Maven directly
mvn spring-boot:run
```

5. Access Application

- **Landing Page**: http://localhost:8080/
- **Admin Dashboard**: http://localhost:8080/admin
- **API Documentation**: http://localhost:8080/swagger-ui.html

Development Setup

```
# Install dependencies
./mvnw clean install

# Run tests
./mvnw test

# Package application
./mvnw package

# Run with specific profile
./mvnw spring-boot:run -Dspring.profiles.active=dev
```

Usage Guide

1. Application Access

1. **Open browser** and navigate to http://localhost:8080
2. **Landing page** displays with login form
3. **Enter credentials** (configure as needed)
4. **Access dashboard** with full functionality

2. Navigation

- **Sidebar**: Main navigation menu
- **Tabs**: Switch between different functional areas
- **Search**: Global search across all modules
- **Theme Toggle**: Switch between light/dark modes

3. Key Operations

Inventory Management

1. **Add Item**: Navigate to Inventory → Add new item
2. **Update Item**: Select item and modify details
3. **Remove Item**: Delete items from inventory
4. **View Inventory**: Browse all items with search/filter

Shipment Handling

1. **Receive Shipment**: Process incoming items
2. **Dispatch Shipment**: Send items to destinations
3. **Track Shipment**: Monitor shipment status
4. **View All**: Complete shipment history

Space Management

1. **View Usage**: Monitor warehouse capacity
2. **Allocate Space**: Assign new storage areas
3. **Free Space**: Release allocated areas

Maintenance

1. **Schedule**: Plan equipment maintenance
2. **Update**: Modify maintenance schedules
3. **Track**: Monitor completion status

Reports

1. **Generate**: Create new reports
2. **View**: Access existing reports
3. **Export**: Download report data

4. API Usage

- **RESTful endpoints** for all operations
- **JSON request/response** format
- **HTTP status codes** for error handling
- **Validation** for data integrity

Development Guidelines

1. Code Standards

- **Java**: Follow Java coding conventions
- **Spring**: Use Spring Boot best practices
- **Database**: Follow JPA/Hibernate patterns
- **Frontend**: Use semantic HTML and modern CSS

2. Architecture Principles

- **Separation of Concerns**: Clear layer separation
- **Single Responsibility**: Each class has one purpose
- **Dependency Injection**: Use Spring's DI container
- **Data Transfer Objects**: Separate DTOs from entities

3. Testing Strategy

- **Unit Tests**: Test individual components
- **Integration Tests**: Test component interactions
- **API Tests**: Test REST endpoints
- **Frontend Tests**: Test UI functionality

4. Security Considerations

- **Input Validation**: Validate all user inputs
- **SQL Injection**: Use parameterized queries
- **Authentication**: Implement proper user authentication
- **Authorization**: Role-based access control

5. Performance Optimization

- **Database Indexing**: Optimize database queries
- **Caching**: Implement appropriate caching strategies
- **Lazy Loading**: Use lazy loading for associations
- **Connection Pooling**: Optimize database connections

6. Error Handling

- **Global Exception Handler**: Centralized error handling
- **Meaningful Error Messages**: User-friendly error descriptions
- **Logging**: Comprehensive logging for debugging
- **Graceful Degradation**: Handle errors gracefully

Troubleshooting

Common Issues

1. Database Connection

Problem: Cannot connect to MySQL **Solution**:

- Verify MySQL service is running
- Check database credentials in `application.properties`

- Ensure database `lwms` exists

2. Port Already in Use

Problem: Port 8080 is already occupied **Solution**:

- Change port in `application.properties`
- Kill process using port 8080
- Use different port number

3. Build Failures

Problem: Maven build errors **Solution**:

- Check Java version (requires Java 21)
- Clean and rebuild: `./mvnw clean install`
- Verify Maven dependencies

4. Frontend Issues

Problem: CSS/JS not loading **Solution**:

- Check static resource paths
- Verify file permissions
- Clear browser cache

Debug Mode

Enable debug logging in `application.properties`:

```
logging.level.com.cts.lwms=DEBUG
logging.level.org.springframework.web=DEBUG
```

Future Enhancements

Planned Features

1. **User Management**: Role-based access control
2. **Advanced Analytics**: Business intelligence dashboard
3. **Mobile App**: Native mobile application
4. **Integration**: ERP system integration
5. **Real-time Updates**: WebSocket notifications
6. **Advanced Reporting**: Custom report builder

Technical Improvements

1. **Microservices**: Break into microservices architecture
2. **Containerization**: Docker deployment

3. **Cloud Deployment**: AWS/Azure integration
4. **Performance**: Redis caching implementation
5. **Security**: OAuth2/JWT authentication

Conclusion

The LWMS system provides a robust foundation for warehouse management operations with its comprehensive feature set, modern architecture, and user-friendly interface. The system is designed to scale with business needs while maintaining code quality and performance standards.

For additional support or questions, refer to the project repository or contact the development team.

Document Version: 1.0
Last Updated: January 2024
Project: Logistics Warehouse Management System (LWMS)
Technology: Spring Boot, Java 21, MySQL, HTML5/CSS3/JavaScript