

# .NET Repository Review & Improvement Plan

---

## 1. Environment & Context

- **Framework:** .NET 8 (LTS)
- **Database:** MySQL 8.x (Implied via Pomelo)
- **Runtime:** Docker / Containerized (Likely K8s or App Service)
- **Auth:** Hybrid (Custom JWT + Azure AD components)
- **Traffic:** High-throughput WMS design patterns observed, but implementation bottlenecks exist.

## 2. Executive Summary

**Overall Health:** ⚠️ **Needs Improvement** The project follows a Clean Architecture structure but suffers from "God Class" service implementations, severe performance anti-patterns (N+1 database writes), and critical security vulnerabilities (hardcoded credentials).

- **Critical Risks:** 2 (Security: Hardcoded Creds, Performance: Loop-DB writes)
- **High Risks:** 3 (JWT Validation, Controller-Service Coupling, Production DB Migration)
- **Code Quality:** God Classes exist in core logic ( StockMovementService ), making maintenance fragility high.

## 3. Impact / Effort Matrix

Priority	Improvement Item	Impact	Effort	Rationale
P0	Remove Hardcoded Credentials	Critical	Low	Active credentials found in TokenMiddleware . Immediate compromise risk.

Priority	Improvement Item	Impact	Effort	Rationale
P0	<b>Fix DB Writes in Loops</b>	Critical	Medium	StockMovementService saves changes inside loops. Will kill DB under load.
P1	<b>Fix JWT Validation</b>	High	Low	IsTokenExpired ignores signature verification. Use standard middleware.
P1	<b>Refactor God Services</b>	High	High	StockMovementService (>2.8k lines) needs splitting by feature/command.
P2	<b>Secure CORS Policy</b>	Medium	Low	AllowAll is open to abuse. Scope to specific subdomains.
P2	<b>Remove EnsureCreated</b>	Medium	Medium	Risk of data loss/schema drift in production. Use EF Migrations.

## 4. Top Improvements

---

### 4.1. Security: Remove Hardcoded Credentials (P0)

**Location:** Shared/Middlewares/TokenMiddleware.cs (Lines 47-50) **Issue:** Plain text username/password/app-key involved in external API calls. **Fix:** Move to Azure Key Vault or Environment Variables.

[object Object]

### 4.2. Performance: Batch Database Writes (P0)

**Location:** Services/Services/Inventory/StockMovementService.cs (Lines 207-241) **Issue:** SaveChangesAsync is called *inside* a foreach loop. For 100 items, this opens 100 DB roundtrips. **Fix:** Add all entities to the context, then call SaveChangesAsync **once** at the end.

[object Object]

### 4.3. Code Quality: Refactor God Class (P1)

**Location:** Services/Services/Inventory/StockMovementService.cs **Issue:** Class handles BinToBin, ItemToBin, validation, and posting. Violates Single Responsibility Principle. **Fix:** Split into dedicated command handlers using MediatR (preferred) or specific services.

- StockMovement.BinToBinService
- StockMovement.PostingService
- StockMovement.ValidationService

## 5. Security & Threat Model

---

- **Auth Vulnerability:** TokenMiddleware.IsTokenExpired (Line 83) manually parses JWTs checking only `exp` claim. **It validates no signature.** An attacker can forge *any* token with a future expiry date and bypass this check if the upstream provider doesn't catch it.
  - *Fix:* Use `Microsoft.AspNetCore.Authentication.JwtBearer` with OIDC discovery.
- **Production Risk:** Program.cs calls `dbContext.Database.EnsureCreated()`. If run against an existing production DB with slight schema drift, it might fail or behave unpredictably. Use `dotnet ef database update` in a CI/CD pipeline or init container.
- **Logging:** RequestResponseLoggingMiddleware logs *all* traffic. Ensure PII/Passwords are redacted from bodies before logging.

## 6. Performance Review

---

- **Caching Strategy:** UserContextMiddleware caches user context, which is good. However, ensure `_cacheService` is distributed (Redis) and not in-memory if scaling to multiple pods, otherwise users will hit "Created" state on every new pod entry.
- **Entity Framework:**
  - **Tracking:** `AsNoTracking()` is correctly used in `UserContextMiddleware`.
  - **Projections:** `Select(x => new { ... })` is used efficiently to fetch only needed columns. Keep this pattern!
  - **Bulk Operations:** You are using `EFCore.BulkExtensions` (seen in csproj). Ensure you use `BulkInsertAsync` / `BulkUpdateAsync` for the high-volume movements instead of standard `AddRangeAsync` for maximum throughput on valid MySQL targets.

## 7. Roadmap

---

### Week 1: Stabilization (The "Bleeding" Phase)

1. [Security] Rotate keys found in code; move to config.
2. [Performance] Fix the loop-save bug in `StockMovementService`.

3. [Security] Enable standard JWT Bearer validation; remove custom `IsTokenExpired`.

## Week 2: Refactoring

1. Extract `TokenMiddleware` logic into a cleaner Authorization Policy or Filter.
2. Split `StockMovementService` into 3 smaller services (Bin, Item, Container operations).
3. Implement EF Core Migrations pipeline.

## Month 1: Modernization

1. Introduce MediatR for Command/Query separation (cleaner Controllers).
2. Shift to Distributed Cache (Redis) for `UserContext` if strictly needed.
3. Comprehensive Unit Tests for the "Core" business logic (currently logic is hard to test due to coupling).

## 8. Suggested PR Plan

---

- **PR #1: "Security patches"**
- **PR #2: "Performance Hotfix - Stock Movement"**
- **PR #3: "Auth Standardization"**

# WMS-API Architecture & Visualizations

---

This document provides a visual breakdown of the WMS-API system, from high-level architecture to specific business logic workflows.

## 1. System Context (C4 Level 1)

---

High-level view of how the WMS API sits within the ecosystem.

[object Object]

## 2. Component Architecture (Onion/Clean)

---

Visualization of the project structure and dependency flow.

[object Object]

## 3. Request Processing Pipeline

---

Shows how a request travels through the middleware chain defined in `Program.cs`.

[object Object]

## 4. Complex Workflow: Bin-to-Container Movement

---

Visualizing the logic inside `StockMovementService.BinToContainerMovement`.

[object Object]

## 5. Deployment View (Docker)

---

Based on `StagingDockerfile`.

[object Object]