



Prompt-Driven Full Stack Application Development with GitHub Copilot, Integrated Standards Checklist, and Code Review

Our SMEs:

- ArchanaV Pawar
- Swapna Kashalkar

Team Members

- Vishnu Priya P
- Kumpatla Sai Chaitanya
- Gokul Siddarth S
- Dheerajsai Tiwari



Introduction:

A robust, full-stack **Inventory Management System** developed using **prompt-driven GitHub Copilot workflows**, designed for local deployment without cloud dependencies. This project demonstrates how AI-assisted development can accelerate delivery while maintaining code quality through structured standards and best practices/standards.

Tech Stack:

- **Frontend:** Angular (TypeScript, HttpClient, Reactive Forms)
- **Backend:** ASP.NET Core Web API (C#)
- **Database:** SQL Server / LocalDB via Entity Framework Core



Problem Statement:

- Manual code reviews are time-consuming and effort-intensive.
- Slow down development velocity and delay releases.
- Prone to human errors, missing coding or security standards.
- Quality depends heavily on reviewer expertise.
- Results in inconsistent quality and standards



Purpose & Key Benefits

- Automate First-Pass Code Reviews
- Accelerated Development Velocity
- Enhanced Code Quality & Security
- Instant Developer Feedback
- Consistent Coding Standards

Existing Manual Process	Proposed System
Developer → WAIT (8-10 Hours) → Manual Review(1-2 Hours) → Slow Feedback Loop	Developer → AI Review (< 2 Minutes) → Developer Fixes → Focused Human Review(~30 Minutes)
Productivity Impact: <ul style="list-style-type: none"> • High Cost: Developer wait time & context switching. • Bottleneck: Senior developer time is blocked. 	Productivity Impact: <ul style="list-style-type: none"> • High Gain: Instant feedback keeps developers in flow. • Efficiency: Senior time is focused only on critical logic.
Total Review Cycle: 1-2 DAYS	Total Review Cycle: < 1 HOUR

Architecture:

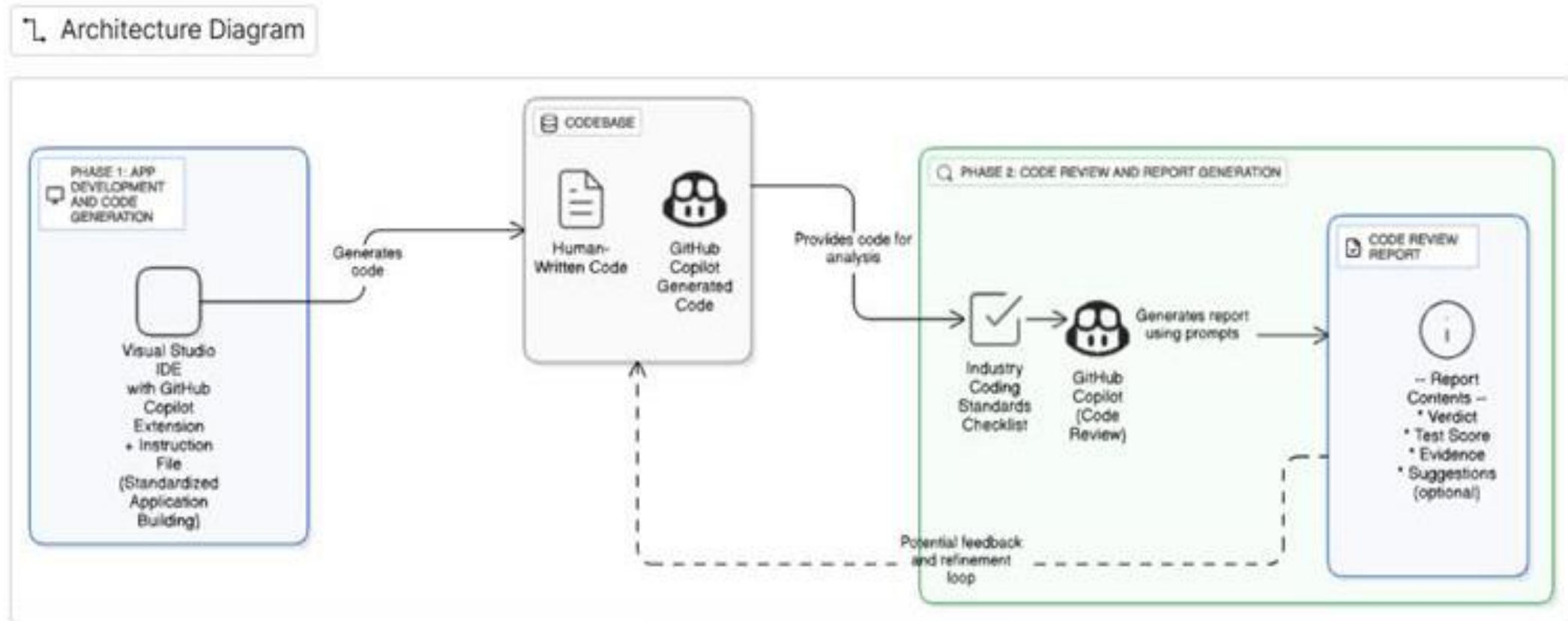


Fig.1 Architecture Diagram for Prompt-Driven App Development and Code Review Workflow Using GitHub Copilot



Project Flow:

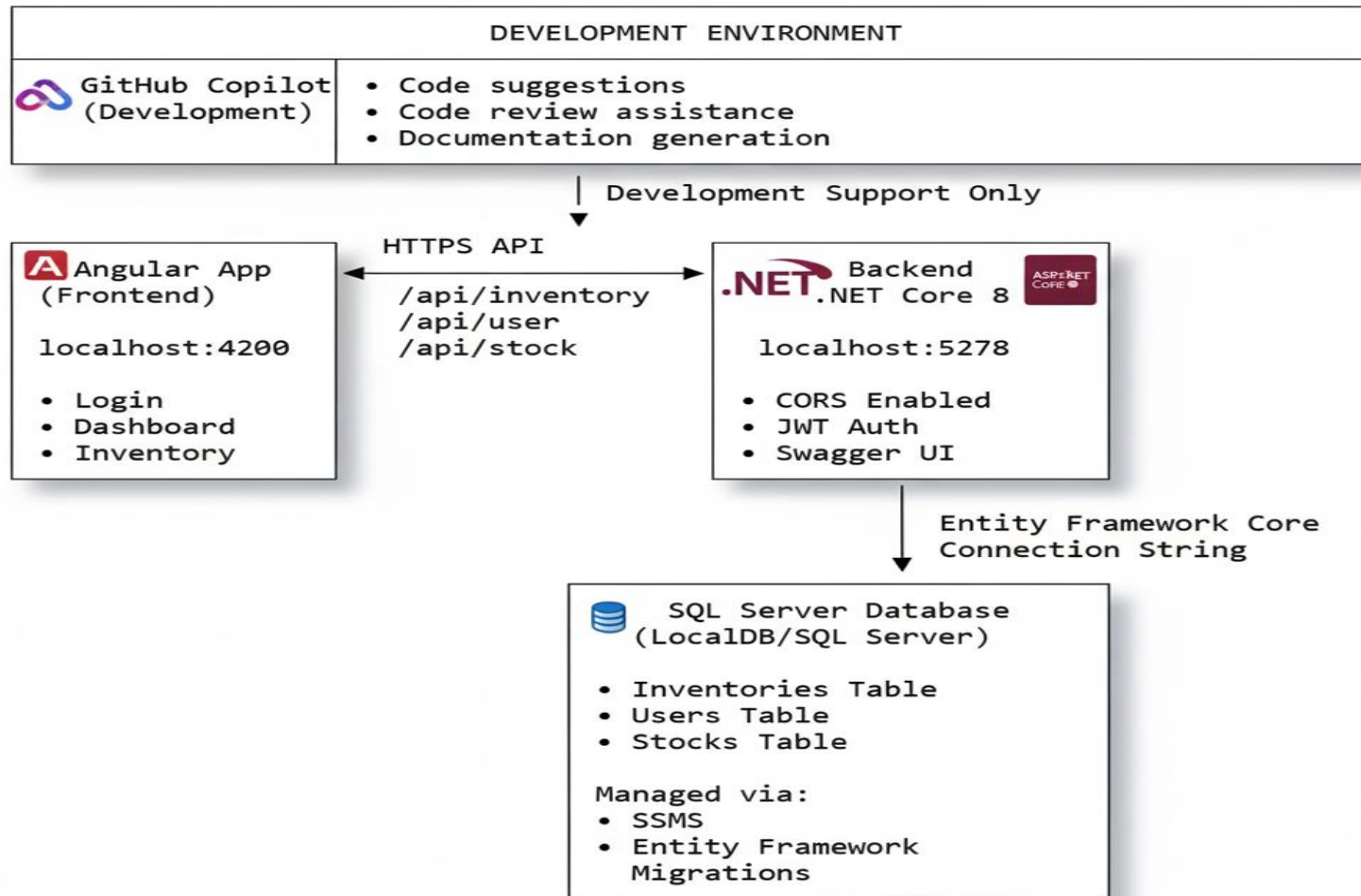


Fig 2: Overview of the development environment and architecture, showing GitHub Copilot integration with Angular frontend, .NET Core 8 backend, and SQL Server database.



AI-Driven Code Review and Prompt Refinement Workflow

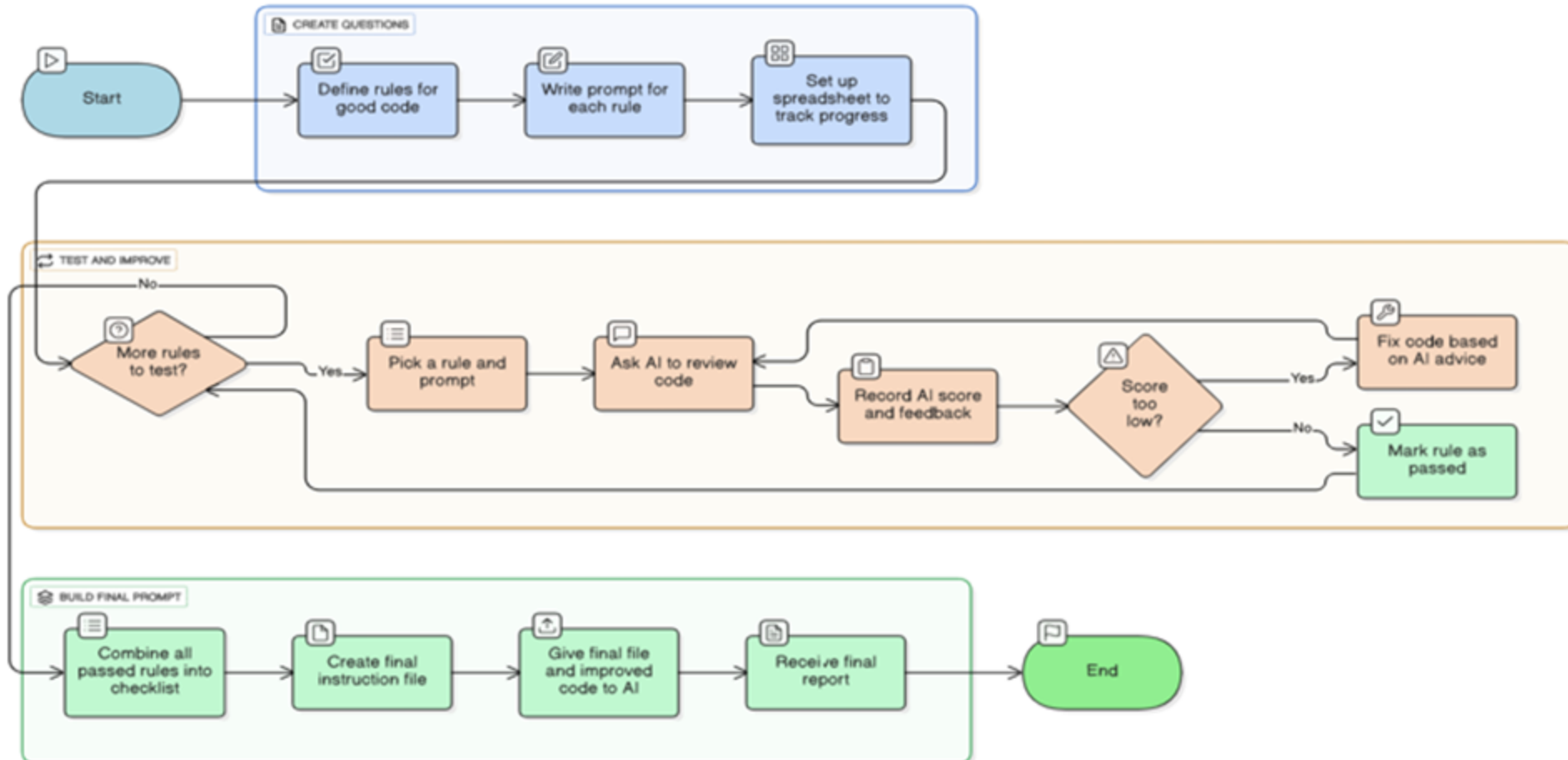
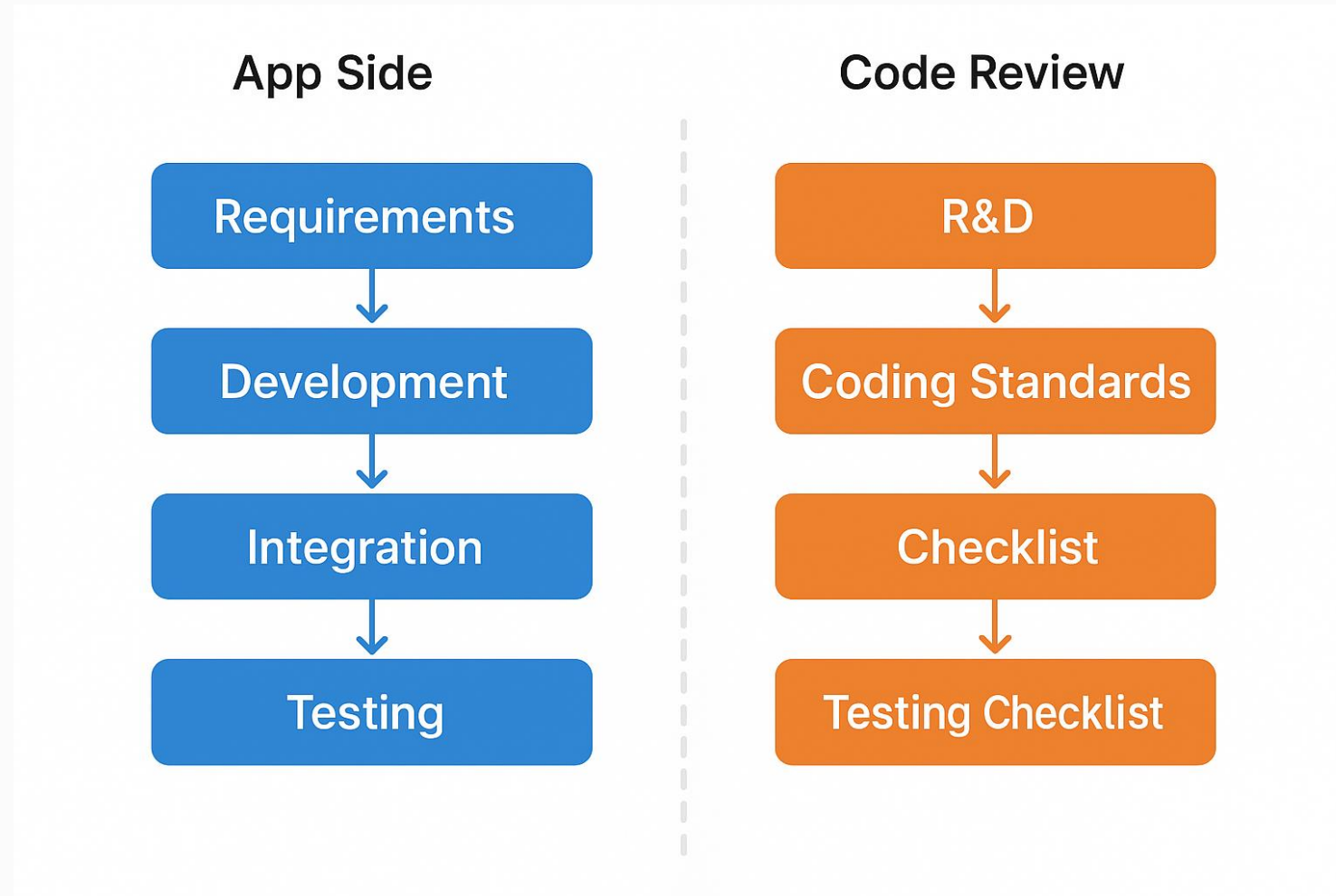


Figure 3: Flowchart depicting the iterative process of code creation, testing, and refinement using AI-driven feedback and rule-based prompts.



Project Phases:





Results :

Front-End Part's Code Review Checklist

CHAT

- Prefer exact file paths and short quotes over long dumps.
- Mark each item Pass/Fail/NA with a one-line reason.
- End with an Overall Verdict and collective score and prioritized suggestions with file/line pointers.

Output:

- Fill the Summary table with ID, Title, Verdict, Test score(1-10), short Notes.
- For each item: Files checked, Evidence, Verdict, Suggestions.
- Keep evidence minimal and targeted.

> Used 1 reference

I'll perform the code review again with test scores and a collective score. Since I already have the file contents from before, I can proceed with generating a new report.

✓

`Get-Date -Format "yyyy-MM-dd_HH-mm-ss"`

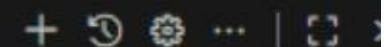
✓

Created 3 todos

Now I'll create the report with test scores for each item and a collective score.



CHAT



- Prefer exact file paths and short quotes over long dumps.
- Mark each item Pass/Fail/NA with a one-line reason.
- End with an Overall Verdict and collective score and prioritized suggestions with file/line pointers.


Output:

- Fill the Summary table with ID, Title, Verdict, Test score(1-10), short Notes.
- For each item: Files checked, Evidence, Verdict, Suggestions.
- Keep evidence minimal and targeted.

> Used 1 reference

I'll perform the code review again with test scores and a collective score. Since I already have the file contents from before, I can proceed with generating a new report.



 `Get-Date -Format "yyyy-MM-dd_HH-mm-ss"`



Created 3 todos

Now I'll create the report with test scores for each item and a collective score.



Back-End Part's Code Review Checklist:

GitHub Copilot Chat

Goal: Perform a comprehensive code quality and security review of the `InventoryManagement.API` ASP.NET Core 8 project. Generate a detailed report summarizing your findings.

Context: You are acting as an SRE (Site Reliability Engineering) agent specialized in .NET code reviews. Your analysis must be strict and adhere to modern best practices for building scalable, secure, and maintainable web APIs. The primary goal is to identify key areas for improvement before the project moves into a production environment.

Source Code to Analyze: Analyze the complete C# source code provided for the `InventoryManagement.API` project, including `Program.cs`, all files in the `Controllers` folder, `Data` folder, and `Models` folder.

Expectations: Your output must be a single, detailed report in Markdown format with the following structure:

1. Executive Summary:

- An overall verdict on the code quality (**Excellent**, **Good**, **Needs Improvement**, or **Critical**).
- An overall numerical score out of 100.
- A summary with the verdict and score for each of the five checklist items below.

2. Detailed Breakdown:

For each of the following five checklist items, provide a detailed analysis section containing:

- Verdict:** Pass or Fail.
- Score:** A numerical score from 1 to 10.
- Analysis & Evidence:** A detailed explanation of your findings, supported by relevant code snippets from the source files.
- Suggestion:** A specific, actionable recommendation for improvement.

Checklist Items for Review:

- A1: Separation of Concerns:** Evaluate if business logic is correctly separated from controller actions. Controllers should be "lean."
- A2: Dependency Injection:** Verify the correct registration and injection of services like `InventoryDbContext`.
- M1: Asynchronous Operations:** Ensure all database and I/O-bound operations are performed asynchronously using `async/await`.
- C1: DTOs for API Contracts:** Assess if `POST/PUT` endpoints use Data Transfer Objects (DTOs) to prevent over-posting vulnerabilities.
- S1: Configuration & Secrets Management:** Analyze how the database connection string is stored and accessed, checking for production readiness.

GitHub Copilot

1 reference

Active Document

Apply

```
# InventoryManagement.API Code Quality & Security Review

## 1. Executive Summary

| Checklist Item | Verdict | Score (/10) |
|---|---|---|
| A1: Separation of Concerns | Pass | 9 |
| A2: Dependency Injection | Pass | 10 |
| M1: Asynchronous Operations | Pass | 10 |
| C1: DTOs for API Contracts | Pass | 10 |
| S1: Configuration & Secrets | Pass | 9 |

**Overall Verdict:** Excellent
**Overall Score:** 96/100


The codebase demonstrates strong adherence to modern .NET best practices for maintainability, scalability, and security. Only minor improvements are recommended for production readiness.

---

## 2. Detailed Breakdown

### A1: Separation of Concerns

- **Verdict:** Pass
- **Score:** 9/10
```

 LTIMindtree

12



Demo



Comparative Study :

Response Time Overview:

- GPT-5: 1.5s initial / 11.2s complete
- Claude Sonnet 3.5: 2.2s initial / 10.8s complete
- Gemini 2.5 Pro: 1.9s initial / 11.5s complete

Performance Leaders:

- Fastest Start: GPT-5 (1.5s)
- Fastest Complete Analysis: Claude (10.8s)
- Most Comprehensive: GPT-5 (9.72/10)

Resource Usage:

- Most Efficient: Claude Sonnet 3.5
- Highest Overhead: GPT-5 (type checking)
- Most Resource-Intensive: Gemini 2.5 Pro (security scanning)

Consistency Metrics:

- Variance between runs: <0.2s
- Linear scaling with codebase size
- Stable performance across code sections



Future Use-Case Enhancements:

- Automated Pull Request Review System
- Role-Based Review Prompts
- Copilot-Driven Code Quality Dashboard
- Integration with Project Management Tools
- Copilot-Powered Documentation Generator



Constraints

- **Slow Azure VM performance** on office hardware hindered development speed
- **Company policy restricted use of personal high-performance PCs**
- **Administrative lockdowns** delayed installations, requiring IT support
- **Stop-and-start workflow** due to dependency on IT availability
- **System restrictions blocked essential software updates and config changes.**
- **Limited Resource Allocation:** VMs may be provisioned with minimal resources, affecting build and test cycles.



Conclusion:

- Delivered a working full-stack inventory system (Angular + ASP.NET Core + SQL Server)
- Frontend-backend integration over HTTPS with CORS and Swagger validation
- Durable data operations verified in SSMS across Inventories, Users, and Stocks
- GitHub Copilot-assisted review rated with strong scores.
- Demo-ready for classrooms and internal tooling.
- Foundation ready for future enhancements: auth hardening, reporting, predictive inventory

THANKYOU