Backend:  
General Code Quality & Best Practices: Checklist Prompts

Category: Architecture & Bootstrap

1. Prompt: Middleware Configuration Order

\*\*Goal:\*\*

Analyze `Program.cs` to verify that the middleware pipeline is configured in the correct and most secure order.

\*\*Context:\*\*

You are an SRE agent reviewing application startup and configuration. In ASP.NET Core, the order in which middleware is registered is critical for security, performance, and functionality. For example, `UseAuthentication` and `UseAuthorization` must be placed correctly relative to `UseCors` and endpoint mapping.

\*\*Source Code to Analyze:\*\*

- `InventoryManagement.API/Program.cs`

\*\*Expectations:\*\*

1. \*\*Verdict:\*\* `Pass` or `Fail`.

2. \*\*Score (1-10):\*\* Rate the correctness of the middleware registration order.

3. \*\*Evidence:\*\* Quote the relevant section of the middleware pipeline from `Program.cs`.

4. \*\*Suggestion:\*\* If `Fail`, provide the corrected order of method calls and explain why the new order is necessary (e.g., "Place `UseCors` before `UseAuthorization` to ensure authorization rules are applied to preflight requests.").

2. Prompt: CORS Policy Security

\*\*Goal:\*\*

Review the Cross-Origin Resource Sharing (CORS) policy defined in `Program.cs` for security best practices.

\*\*Context:\*\*

You are a security review agent. The CORS policy is a critical security feature that prevents unauthorized web pages from making requests to the API. The policy should be as restrictive as possible, allowing requests only from known origins and with specific methods/headers, rather than using overly permissive settings like `AllowAnyOrigin()`.

\*\*Source Code to Analyze:\*\*

- `InventoryManagement.API/Program.cs`

\*\*Expectations:\*\*

1. \*\*Verdict:\*\* `Pass` or `Fail`.

2. \*\*Score (1-10):\*\* Rate the security and specificity of the CORS policy.

3. \*\*Evidence:\*\* Provide the code snippet defining the CORS policy.

4. \*\*Suggestion:\*\* If the policy is too permissive, recommend specific changes, such as replacing `AllowAnyHeader()` with `WithHeaders(...)` that lists only the required headers.

Category: Models & Types

1. Prompt: Model Validation Consistency

\*\*Goal:\*\*

Audit all classes in the `Models` folder to ensure consistent and appropriate use of Data Annotations for validation.

\*\*Context:\*\*

You are a data integrity agent. To ensure data quality at the earliest stage, all model properties should be decorated with appropriate validation attributes like `[Required]`, `[StringLength]`, `[Range]`, and `[EmailAddress]`. This helps enforce business rules and prevents invalid data from reaching the database.

\*\*Source Code to Analyze:\*\*

- All files in `InventoryManagement.API/Models/`

\*\*Expectations:\*\*

1. \*\*Verdict:\*\* `Pass` or `Fail`.

2. \*\*Score (1-10):\*\* Rate the completeness of the model validation.

3. \*\*Evidence:\*\* Provide an example of a model property that is well-validated or one that is missing necessary validation.

4. \*\*Suggestion:\*\* If validation is missing, specify the model and property, and recommend the appropriate Data Annotation to add (e.g., "Add `[StringLength(50)]` to the `UserType` property in `User.cs`.").

4. Prompt: EF Core Relationships & Naming:

\*\*Goal:\*\*

Verify that relationships between EF Core models are correctly defined and that primary/foreign key naming conventions are consistent.

\*\*Context:\*\*

You are a database schema review agent. Well-defined relationships (one-to-many, many-to-many) using navigation properties and foreign key attributes are essential for data integrity. Furthermore, inconsistent naming for keys (e.g., mixing `Id`, `ID`, and `InventoryID`) can lead to confusion and errors.

\*\*Source Code to Analyze:\*\*

- All files in `InventoryManagement.API/Models/`

- `InventoryManagement.API/Data/InventoryDbContext.cs`

\*\*Expectations:\*\*

1. \*\*Verdict:\*\* `Pass` or `Fail`.

2. \*\*Score (1-10):\*\* Rate the correctness and consistency of the data model.

3. \*\*Evidence:\*\* Quote a correctly defined relationship or an example of inconsistent naming.

4. \*\*Suggestion:\*\* If issues are found, recommend specific changes, such as "Standardize all primary keys to the format `{ModelName}ID`" or "Add a `[ForeignKey("UserID")]` attribute to the `Stock` model."

Category: Services & Data Flow

5. Prompt: Efficient Database Queries

\*\*Goal:\*\*

Analyze the database queries in `InventoryController.cs` to identify potential performance issues, such as the "N+1 problem" or inefficient data retrieval.

\*\*Context:\*\*

You are a performance optimization agent. When retrieving related data, developers should use methods like `Include()` and `ThenInclude()` to eagerly load data in a single query. Failing to do so can result in numerous round-trips to the database (the N+1 problem), severely degrading performance.

\*\*Source Code to Analyze:\*\*

- `InventoryManagement.API/Controllers/InventoryController.cs`

\*\*Expectations:\*\*

1. \*\*Verdict:\*\* `Pass` or `Fail`.

2. \*\*Score (1-10):\*\* Rate the efficiency of the database queries.

3. \*\*Evidence:\*\* Provide a code snippet of a query.

4. \*\*Suggestion:\*\* If a query is inefficient, provide the optimized version. For example, change `\_context.Inventories.ToListAsync()` to `\_context.Inventories.Include(i => i.Stocks).ToListAsync()` if the stocks are always needed.

6. Prompt: Service Layer Abstraction:

\*\*Goal:\*\*

Re-evaluate `InventoryController.cs` to confirm that it contains no direct `DbContext` access and instead delegates all data operations to an abstracted service layer.

\*\*Context:\*\*

You are an architectural review agent. As a follow-up to the initial review, this prompt enforces the rule that controllers must not interact directly with `DbContext`. All data logic must be encapsulated within a service (e.g., `IInventoryService`). This makes the code more testable, maintainable, and modular.

\*\*Source Code to Analyze:\*\*

- `InventoryManagement.API/Controllers/InventoryController.cs`

\*\*Expectations:\*\*

1. \*\*Verdict:\*\* `Pass` or `Fail`.

2. \*\*Score (1-10):\*\* 10 if the controller is completely free of `DbContext` references.

3. \*\*Evidence:\*\* Show the controller's constructor. It should inject a service interface, not `InventoryDbContext`.

4. \*\*Suggestion:\*\* If `DbContext` is still present, reiterate the need to refactor this logic into a separate service class and inject the service into the controller.

Category: Error Handling

7. Prompt: Global Exception Handling:

\*\*Goal:\*\*

Check `Program.cs` for the presence of a global exception handling middleware.

\*\*Context:\*\*

You are a reliability agent. To prevent sensitive application details from leaking in unhandled exceptions and to provide a consistent error response format to clients, a global exception handler is a best practice. This middleware should catch any unhandled exceptions and log them while returning a generic, user-friendly error message.

\*\*Source Code to Analyze:\*\*

- `InventoryManagement.API/Program.cs`

\*\*Expectations:\*\*

1. \*\*Verdict:\*\* `Pass` or `Fail`.

2. \*\*Score (1-10):\*\* Rate the implementation of global error handling.

3. \*\*Evidence:\*\* Quote the code that registers the exception handling middleware (e.g., `app.UseExceptionHandler(...)`).

4. \*\*Suggestion:\*\* If no global handler is present, recommend adding one, such as: "In `Program.cs`, add `app.UseExceptionHandler(\"/error\")` and create a corresponding error handling endpoint."

8. Prompt: Specific Exception Handling

\*\*Goal:\*\*

Analyze the `try-catch` blocks within `InventoryController.cs` to ensure they handle specific, expected exceptions rather than catching a generic `Exception`.

\*\*Context:\*\*

You are a code correctness agent. Catching a generic `System.Exception` is often a bad practice because it can hide programming errors and make debugging difficult. Code should catch specific exceptions it can meaningfully handle, such as `DbUpdateConcurrencyException` or `InvalidOperationException`.

\*\*Source Code to Analyze:\*\*

- `InventoryManagement.API/Controllers/InventoryController.cs`

\*\*Expectations:\*\*

1. \*\*Verdict:\*\* `Pass` or `Fail`.

2. \*\*Score (1-10):\*\* Rate the specificity and correctness of the exception handling.

3. \*\*Evidence:\*\* Provide a `try-catch` block from the source code.

4. \*\*Suggestion:\*\* If a generic `catch (Exception e)` is found, recommend replacing it with a catch block for a more specific exception type relevant to the code within the `try` block.

Category: .NET Project & Build

9. Prompt: Project File (.csproj) Analysis

\*\*Goal:\*\*

Review the `InventoryManagement.API.csproj` file for obsolete package references, correct .NET version, and nullable settings.

\*\*Context:\*\*

You are a project maintenance agent. The `.csproj` file defines the project's dependencies and build settings. It's important to ensure it targets a current .NET version (e.g., .NET 8), does not contain unused or vulnerable NuGet packages, and has modern C# features like `Nullable` enabled to improve code quality.

\*\*Source Code to Analyze:\*\*

- `InventoryManagement.API/InventoryManagement.API.csproj`

\*\*Expectations:\*\*

1. \*\*Verdict:\*\* `Pass` or `Fail`.

2. \*\*Score (1-10):\*\* Rate the health and configuration of the project file.

3. \*\*Evidence:\*\* Quote relevant sections from the `.csproj` file, such as `<TargetFramework>` or `<Nullable>`.

4. \*\*Suggestion:\*\* If issues are found, recommend specific changes like "Update the `TargetFramework` to `net8.0`" or "Enable nullability by setting `<Nullable>enable</Nullable>` to prevent null reference exceptions."

10. Prompt: Environment Configuration

\*\*Goal:\*\*

Verify that the application correctly uses environment-specific configurations, such as enabling Swagger only for the Development environment.

\*\*Context:\*\*

You are a deployment readiness agent. For security and correctness, certain application behaviors should differ between Development, Staging, and Production environments. This is typically controlled by checking `IWebHostEnvironment.IsDevelopment()` (or similar) and loading settings from `appsettings.{Environment}.json`.

\*\*Source Code to Analyze:\*\*

- `InventoryManagement.API/Program.cs`

\*\*Expectations:\*\*

1. \*\*Verdict:\*\* `Pass` or `Fail`.

2. \*\*Score (1-10):\*\* Rate the implementation of environment-specific logic.

3. \*\*Evidence:\*\* Quote the `if (app.Environment.IsDevelopment())` block.

4. \*\*Suggestion:\*\* If production-unsafe features (like detailed error pages or Swagger UI) are not conditionally enabled, recommend wrapping them in an `if (app.Environment.IsDevelopment())` block.

Checklist md prompt:  
**Prompt for Code Review Checklist:**

**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.

Angular Frontend: Code Quality & Best Practices Checklist Prompts

Category: Architecture & Component Design

1. Prompt: Separation of Concerns (SRP)

\*\*Goal:\*\*

Analyze the provided Angular component file to determine if it adheres to the Single Responsibility Principle (SRP).

\*\*Context:\*\*

You are an Angular architecture review agent. A "lean" component should only be responsible for UI and presentation logic. It must delegate all business logic, data fetching, state management, and direct data manipulation to injectable services. Evaluate if the component's role is strictly limited to managing its template and view state.

\*\*Source Code to Analyze:\*\*

- `src/app/features/inventory/components/inventory-list/inventory-list.component.ts`

\*\*Expectations:\*\*

1. \*\*Verdict:\*\* `Pass` or `Fail`.

2. \*\*Score (1-10):\*\* Rate the component's adherence to SRP, where 10 is a perfectly lean component.

3. \*\*Evidence:\*\* Provide a code snippet that shows either a clean delegation to a service (Pass) or business logic being handled directly within the component (Fail).

4. \*\*Suggestion:\*\* If `Fail`, recommend refactoring the identified business logic into a dedicated service (e.g., `InventoryService`) and injecting that service into the component.

2. Prompt: Naming Conventions

\*\*Goal:\*\*

Audit the project's file and symbol names for adherence to the official Angular style guide.

\*\*Context:\*\*

You are a code consistency agent. The Angular style guide prescribes specific naming conventions for maintainability:

- File names must be lowercase and use kebab-case (e.g., `inventory-list.component.ts`).

- Class and symbol names should follow the `[Feature].[Type]` pattern and use PascalCase with the correct suffix (e.g., `InventoryListComponent`, `AuthGuard`, `InventoryService`).

\*\*Source Code to Analyze:\*\*

- Provide the file structure of a feature module (e.g., `src/app/features/inventory/`).

- `src/app/features/inventory/components/inventory-list/inventory-list.component.ts`

- `src/app/features/inventory/services/inventory.service.ts`

\*\*Expectations:\*\*

1. \*\*Verdict:\*\* `Pass` or `Fail`.

2. \*\*Score (1-10):\*\* Rate the overall consistency of naming conventions.

3. \*\*Evidence:\*\* List any file or class names that violate the conventions.

4. \*\*Suggestion:\*\* Provide a specific list of files or classes that need to be renamed to conform to the style guide.

3. Prompt: Template Cleanliness (No Logic)

\*\*Goal:\*\*

Review the component's HTML template to ensure it is purely declarative and free of complex logic.

\*\*Context:\*\*

You are a template review agent. Angular templates should be simple and readable. Any complex logic, such as chained conditions (`\*ngIf="a && b && c"`), complex arithmetic, or function calls with multiple arguments, should be moved into the component's class. The template should bind to simple properties or call simple getter methods.

\*\*Source Code to Analyze:\*\*

- `src/app/features/inventory/components/inventory-list/inventory-list.component.html`

\*\*Expectations:\*\*

1. \*\*Verdict:\*\* `Pass` or `Fail`.

2. \*\*Score (1-10):\*\* Rate the cleanliness and declarativeness of the template.

3. \*\*Evidence:\*\* Quote a line from the template that contains complex logic.

4. \*\*Suggestion:\*\* If `Fail`, recommend moving the logic into a new method or property on the component's `.ts` file and updating the template to bind to the new member.

Category: Performance & Memory Management

1. Prompt: RxJS and Memory Management

\*\*Goal:\*\*

Audit the component for potential memory leaks by checking for unmanaged RxJS subscriptions.

\*\*Context:\*\*

You are a memory management agent. Manually subscribing to an Observable (`.subscribe(...)`) without a proper teardown mechanism is a common source of memory leaks in Angular. The best practice is to use the `async` pipe in the template. If manual subscription is necessary, an explicit unsubscribe mechanism (like `takeUntil` paired with `ngOnDestroy`) must be implemented.

\*\*Source Code to Analyze:\*\*

- `src/app/features/inventory/components/inventory-details/inventory-details.component.ts`

\*\*Expectations:\*\*

1. \*\*Verdict:\*\* `Pass` or `Fail`.

2. \*\*Score (1-10):\*\* Rate the safety of the RxJS subscription handling.

3. \*\*Evidence:\*\* Provide a code snippet showing a `.subscribe()` call that is not properly unsubscribed in `ngOnDestroy`.

4. \*\*Suggestion:\*\* If a potential leak is found, recommend either using the `async` pipe in the template or implementing the `takeUntil(this.destroy$)` pattern.

1. Prompt: Immutability & Change Detection:

\*\*Goal:\*\*

Verify that the component uses `ChangeDetectionStrategy.OnPush` and treats data inputs as immutable.

\*\*Context:\*\*

You are a performance optimization agent. For better performance, presentational components should use the `OnPush` change detection strategy. This requires that all `@Input()` data is treated as immutable. When data needs to be updated, a new object or array reference must be created instead of mutating the existing one in place.

\*\*Source Code to Analyze:\*\*

- `src/app/shared/components/button/button.component.ts` (as an example of a presentational component)

\*\*Expectations:\*\*

1. \*\*Verdict:\*\* `Pass` or `Fail`.

2. \*\*Score (1-10):\*\* Rate the implementation of `OnPush` and immutability.

3. \*\*Evidence:\*\* Quote the `@Component` decorator and any code that directly mutates an `@Input()` property.

4. \*\*Suggestion:\*\* If not present, recommend adding `changeDetection: ChangeDetectionStrategy.OnPush` to the component decorator and replacing any data mutations with immutable patterns (e.g., using the spread operator `...`).

1. Prompt: Performance Optimizations (trackBy)

\*\*Goal:\*\*

Check if all `\*ngFor` loops that iterate over lists of objects are optimized with a `trackBy` function.

\*\*Context:\*\*

You are a DOM performance agent. When an array is modified, Angular re-renders the entire DOM list by default. Providing a `trackBy` function tells Angular how to track each item, allowing it to create or destroy only the DOM elements that have changed. This is a critical performance optimization for large or frequently updated lists.

\*\*Source Code to Analyze:\*\*

- `src/app/features/inventory/components/inventory-list/inventory-list.component.html`

\*\*Expectations:\*\*

1. \*\*Verdict:\*\* `Pass` or `Fail`.

2. \*\*Score (1-10):\*\* Rate the usage of `trackBy` for list rendering.

3. \*\*Evidence:\*\* Quote an `\*ngFor` loop that is missing a `trackBy` function.

4. \*\*Suggestion:\*\* If `trackBy` is missing, provide a sample `trackBy` function for the component class and show how to add `: trackBy inventoryTrackBy` to the `\*ngFor` expression in the template.

Full checklist:  
\*\*Goal:\*\*

Perform a comprehensive code quality and best practices review of the entire Angular frontend project. Generate a single, detailed report summarizing your findings.

\*\*Context:\*\*

You are acting as an SRE (Site Reliability Engineering) agent specialized in modern Angular development. Your analysis must be strict and adhere to the official Angular style guide and best practices for building scalable, performant, and maintainable Single-Page Applications (SPAs). The primary goal is to identify key areas for improvement in architecture, performance, and code quality.

\*\*Source Code to Analyze:\*\*

Analyze the complete TypeScript and HTML source code within the `src/app/` directory of the Angular project. Assume a standard Angular CLI project structure with feature modules.

\*\*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 table with the verdict and score for each of the ten checklist items below.

2. \*\*Detailed Breakdown:\*\*

For each of the following ten 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.

\* \*\*Suggestion:\*\* A specific, actionable recommendation for improvement.

\*\*Checklist Items for Review:\*\*

\* \*\*1. Separation of Concerns (SRP):\*\* Components should be lean, delegating business logic to services.

\* \*\*2. Naming Conventions:\*\* Files (`kebab-case`) and symbols (`PascalCase` with suffixes) must follow the Angular style guide.

\* \*\*3. Template Cleanliness (No Logic):\*\* Templates must be declarative, with complex logic moved to the component class.

\* \*\*4. RxJS and Memory Management:\*\* Observables must be properly unsubscribed (e.g., `async` pipe, `takeUntil`) to prevent memory leaks.

\* \*\*5. TypeScript Type Safety:\*\* The `any` type must be avoided, and strict mode should be enabled.

\* \*\*6. Immutability & Change Detection:\*\* `ChangeDetectionStrategy.OnPush` should be used where appropriate, and data treated as immutable.

\* \*\*7. Performance Optimizations:\*\* `trackBy` function must be used with `\*ngFor` for object arrays.

\* \*\*8. Code Structure & DRY:\*\* The project should have a logical feature-based structure, and duplicated code must be avoided.

\* \*\*9. Function and Line Limits:\*\* Files and functions should be concise and purposeful (e.g., function < 75 lines, file < 400 lines).

\* \*\*10. Dependency Injection Scope:\*\* Services must be provided at the correct scope (e.g., `providedIn: 'root'` for singletons).

Agent prompt:

can you do this changes and rerun the above prompt and check the scores

template for agentic prompt:  
\*\*Goal:\*\*

Analyze the provided [Framework/Language] code to evaluate its adherence to the "[Checklist Item Title]" principle.

\*\*Context:\*\*

You are acting as a [Specify Agent Persona, e.g., Code Quality Agent, Security Review Agent, Performance Optimization Agent]. The project standard requires that [Describe the Best Practice or Rule to Enforce]. For example, [Provide a specific example of the rule, e.g., "business logic must be delegated to services, not handled in components."].

\*\*Source Code to Analyze:\*\*

```[language]

// PASTE THE FULL CONTENT OF YOUR

// [File Path, e.g., src/app/features/dashboard/dashboard.component.ts]

// FILE(S) HERE.

Expectations:

Verdict: Pass or Fail.

Score (1-10): Rate the code's adherence to this specific best practice.

Evidence: Provide a direct code snippet from the source code that supports your verdict.

Suggestion: If the verdict is Fail, provide a specific, actionable recommendation for how to fix the code to meet the standard.

#### \*\*How to Fill in Template 1:\*\*

\* \*\*`[Framework/Language]`\*\*: `Angular`, `ASP.NET Core`, `TypeScript`, `C#`, etc.

\* \*\*`[Checklist Item Title]`\*\*: A short name for the rule, e.g., `Separation of Concerns`, `Immutable State`, `Secure Headers`.

\* \*\*`[Specify Agent Persona]`\*\*: The role the AI should take, e.g., `Architecture Review Agent`, `Security Analyst`.

\* \*\*`[Describe the Best Practice or Rule to Enforce]`\*\*: Clearly explain the standard you are checking against.

\* \*\*`[Provide a specific example of the rule]`\*\*: Give a concrete example to remove ambiguity.

\* \*\*`[language]`\*\*: `typescript`, `html`, `csharp`, etc., for syntax highlighting.

\* \*\*`[File Path]`\*\*: The path to the file you are pasting, for context.

---

### \*\*Template 2: Comprehensive Report Generation Prompt\*\*

Use this template to ask the agent to perform a full review of your application against a list of predefined checklist items and generate a single, consolidated report.

```markdown

\*\*Goal:\*\*

Perform a comprehensive code quality and best practices review of the entire [Frontend/Backend] project. Generate a single, detailed report summarizing your findings.

\*\*Context:\*\*

You are acting as an SRE (Site Reliability Engineering) agent specialized in modern [Framework/Language] development. Your analysis must be strict and adhere to best practices for building scalable, performant, and maintainable applications. You will be analyzing the full source code provided below.

\*\*Source Code to Analyze:\*\*

```[language]

// PASTE THE CONTENT OF ALL RELEVANT FILES HERE.

// TO GET A COMPLETE AND ACCURATE REVIEW, INCLUDE ALL FILES

// RELATED TO THE CHECKLIST ITEMS BELOW.

Expectations:

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

Executive Summary:

An overall verdict on the code quality (Excellent, Good, Needs Improvement, or Critical).

An overall numerical score out of 100.

A summary table with the verdict and score for each checklist item.

Detailed Breakdown:

For each of the checklist items listed below, provide a detailed analysis section containing:

Verdict: Pass or Fail.

Score: A numerical score from 1 to 10.

Analysis & Evidence: A detailed explanation supported by code snippets.

Suggestion: A specific, actionable recommendation for improvement.

Checklist Items for Review:

1. [Title of Checklist Item 1]: [Short description of the rule].

2. [Title of Checklist Item 2]: [Short description of the rule].

3. [Title of Checklist Item 3]: [Short description of the rule].

4. [Title of Checklist Item 4]: [Short description of the rule].

5. [Title of Checklist Item 5]: [Short description of the rule].

...[Add as many items as you need]

#### \*\*How to Fill in Template 2:\*\*

\* \*\*`[Frontend/Backend]`\*\*: Specify which part of the application is being reviewed.

\* \*\*`[Framework/Language]`\*\*: The main technology stack, e.g., `Angular`, `ASP.NET Core`.

\* \*\*`[language]`\*\*: `typescript`, `csharp`, etc.

\* \*\*`[Title of Checklist Item X]`\*\*: The short name for each rule (e.g., `Separation of Concerns`).

\* \*\*`[Short description of the rule]`\*\*: A brief explanation of the standard for that]

**Helper prompt:**

* Can you create a docs folder and keep generated responses in reports, prompts used in one subfolder, and checklists in one subfolder.
* Then generate corresponding md files in the respective folders