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 i

**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
```

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 req

**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 `WithHead
```

Category: Models & Types

3. 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 appropri

**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 (
```

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 consisten

**Context:**
You are a database schema review agent. Well-defined relationships (one-to-many, many-to-many) using navigation properties and forei

**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
```

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 inef

**Context:**
You are a performance optimization agent. When retrieving related data, developers should use methods like `Include()` and `ThenIncl

**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(
```

6. Prompt: Service Layer Abstraction

```
**Goal:**
Re-evaluate `InventoryController.cs` to confirm that it contains no direct `DbContext` access and instead delegates all data operati

**Context:**
You are an architectural review agent. As a follow-up to the initial review, this prompt enforces the rule that controllers must not

**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 inj
```

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 consiste

**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(\
```

8. Prompt: Specific Exception Handling

```
**Goal:**
Analyze the `try-catch` blocks within `InventoryController.cs` to ensure they handle specific, expected exceptions rather than catch

**Context:**
You are a code correctness agent. Catching a generic `System.Exception` is often a bad practice because it can hide programming erro

**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 excep
```

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 ens

**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 nulla
```

10. Prompt: Environment Configuration

```
**Goal:**
Verify that the application correctly uses environment-specific configurations, such as enabling Swagger only for the Development en

**Context:**
You are a deployment readiness agent. For security and correctness, certain application behaviors should differ between Development,

**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
```