**C# Coding Standards and Naming Conventions**

Sept 2021

Content

[Software Architecture 2](#_Toc87538351)

[• Independent from third-party bookstores 2](#_Toc87538352)

[• Testable 2](#_Toc87538353)

[• Independent of sight 2](#_Toc87538354)

[• Independent of database technology 2](#_Toc87538355)

[• Independent from outside agents 2](#_Toc87538356)

[Common and Typical abstractions in an architecture 3](#_Toc87538357)

[Clean Architecture N Layers 4](#_Toc87538358)

[Technological stack used in this Test 7](#_Toc87538359)

[Unit Test, Swagger & Postman 9](#_Toc87538360)

# 

# Software Architecture

In this test: (CMG Engineering Audition) I have used CLEAN ARCHITECTURE, based on the software design principle: Separation of responsibilities. In this type of software architecture proposed in this Test, the layers are established, each with a specific responsibility. More information can be found in **Uncle Bob's** article. Thanks to this separation, systems are achieved:

## • Independent from third-party bookstores

## • Testable

## • Independent of sight

## • Independent of database technology

## • Independent from outside agents

**Why I choose a clean architecture**

A clean architecture like used in this Test (CMG Engineering Audition) allows me these benefits for that reason I had chosen:

1. Iterate / evolve your product faster. Each module takes responsibility for one or more related use cases. Is there a problem in a specific module? The problem is simply searched for in the files related to that module (specifically in its tests).

2. Activating and deactivating functionalities and modules is faster. By having all the files of a module located in a single point in your project, the activation and deactivation of certain functionalities is faster (it can even be done with a flag on the server).

3. Enabling unit test, and code analysis techniques, allows:

Gain confidence. After each change uploaded to the server, the technical manager feels supported, knowing that the new changes do not cause failures in what was already working.

Find and fix bugs faster. A test case should be the functional specification of a module. If there is a problem in a module, the first place to look is the files and folders in that test case.

Establish regressions. Tests can be added as errors are found in the code to make sure they don't happen again.

4. Achieve a more manageable, maintainable and flexible code in the long term.

5. There are clear standards and conventions on how to program, but the architectures set the guidelines in which a programmer must adhere to introduce new functionalities in the application. It is relatively easy to get a new developer to learn these guidelines if you have a defined architecture.

6. Easily allows an application to evolve, (solution) initially created as Monolithic to various solutions focused on microservices, migrating specific modules (use cases) to these new solutions, even considering different alternatives in data handling, and front end.

7. Immediate implementation

It can be implemented with any programming language, among which we mention: Java, .Net, Php, Node.js.

8. The primary focus of the project is placed on the core and logic of the domain.

9. This architecture allows important changes to the application, without major impacts:

You could change the framework used, if necessary, since everything is decoupled, you could replace the database or add some other if needed.

10. The Result is an optimal, solid, quality, and scalable product.

## Common and Typical abstractions in an architecture

Imagen de la pantalla de un celular con letras

Descripción generada automáticamente con confianza media

Plato con pasta y vegetales

Descripción generada automáticamente The way to solve these abstractions can lead us to an architecture like this:

• Complex

• Inconsistent

• Rigid

• Not maintainable

• Not testable

Comida en un plato

Descripción generada automáticamente It is suggested in this guide to reach a clear level of abstraction, as in the following graphic:

• Simple

• Understandable

• Flexible

• Maintainable

• Testable

• Scalable

## Clean Architecture N Layers

Un pastel decorado

Descripción generada automáticamente con confianza media The clean architecture presents distribution of software components, where logical levels with different roles are distinguished enclosed in rings in the following graphic:

Gráfico, Gráfico de proyección solar

Descripción generada automáticamente

Each color is:

a unique responsibility.

a different level of abstraction.

Different developer roles

**Explanation of each layer:**

* **Domain** - Contains types and business logic
* Application - Contains business logic and types
* **Infrastructure**: (including persistence) contains all external abstractions, and transversal to development
* **Presentation**: It is the visual presentation layer.

Diagrama

Descripción generada automáticamente con confianza media

Graphic explanation. Infrastructure and presentation components can be replaced with minimal effort.

**Gráfico, Gráfico de proyección solar

Descripción generada automáticamenteDomain**:

It has the Entities, Value Objects, Enumerations and Logical Exceptions, Infrastructure accesses this layer and IoC (Inversion of Control) will help us to inject dependencies. In this test I had a AuditEntity class mapped to a database table to save all the readings of each devices, also a AuditLog class to use with NLog

Imagen de la pantalla de un celular con letras

Descripción generada automáticamente con confianza baja **Application**:

This layer has the interfaces, models, logical commands and queries, validations and exceptions, it is the level of logical abstraction of the application that solves the Use cases. It implements the principle of single responsibility, multiple implementations. I have her some DTOS files (Audition devices class, Mail Dtos, etc. The Feature folder has the code to Evaluate the Log on EvaluateLogCommand class. This class will call EvaluateLogFile method from the IAuditionrepository interface.

Gráfico, Gráfico de proyección solar

Descripción generada automáticamente **Persistence**:

This layer has the DbContext, migrations, configurations, initial data seeding, and data abstractions. In this project is located on CMGEngineeringAudition.Infrastructure project with a context database to save some info (Not mandatory in this test, but necessary in real life), also has the AuditionRepository class to read the file with all the measurements

Gráfico, Gráfico de proyección solar

Descripción generada automáticamente **Infrastructure**:

This layer implements Inversion of Control, Implementations as external APIs, File System, Email / SMS, any external abstraction, or transversal to the system. In this case I have created a services folder to store services class to make math calculations, send mails, evaluate expressions. For this reason, there is a ListService, MathService, NumberService, etc.

Imagen de la pantalla de un celular con letras

Descripción generada automáticamente con confianza baja **Presentation**:

This layer contains the presentation layer developed in MVC .NET CORE Application, Includes a single controller, and application startup. QualityControlController use MediatR to call a Command features in the application layer and return the json file.

## Technological stack used in this Test

• ASP.NET Core 5.0 WebAPI

• Entity Framework Core 5.0

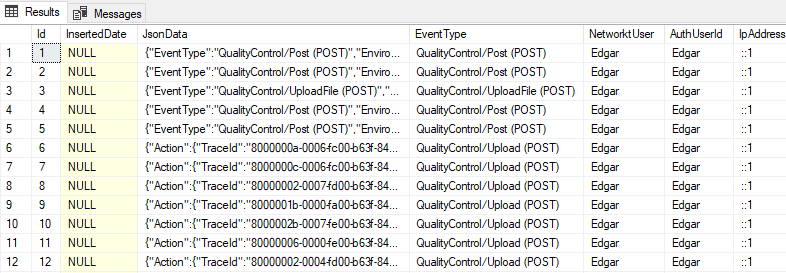
**Open Source:** This Project will mainly use nugget packages licensed under the MIT license.

**Features**

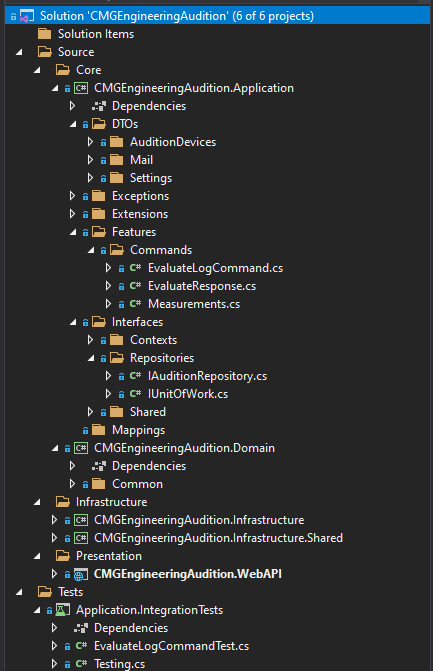
* Clean architecture
* CQRS with MediatR library
* Entity Framework Core: code first
* Repository master
* MediatR Pipeline Logging, Validation NLOG and Serilog
* Swagger UI
* Response Wrappers, Pagination
* Custom exception handling middleware
* API version control
* Fluent Validation and Automapper

**Solution proposal in Visual Studio**

The proposal application create a Log file in the Database: *CMG\_AuditionServices* in the table **AuditLog**, also has the necessary classes to send mail one the audit log has been Done

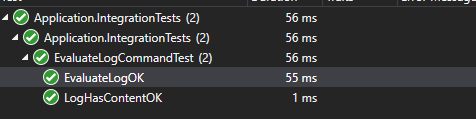


Data sample

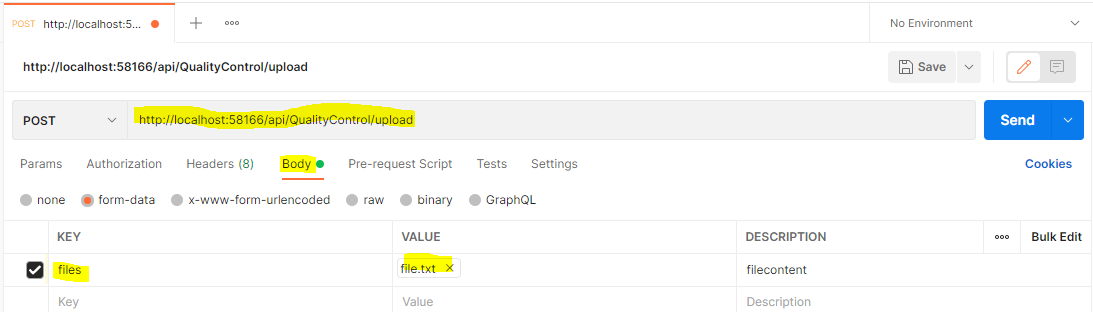


## Unit Test, Swagger & Postman

The solution has a unit test project with 2 test done:



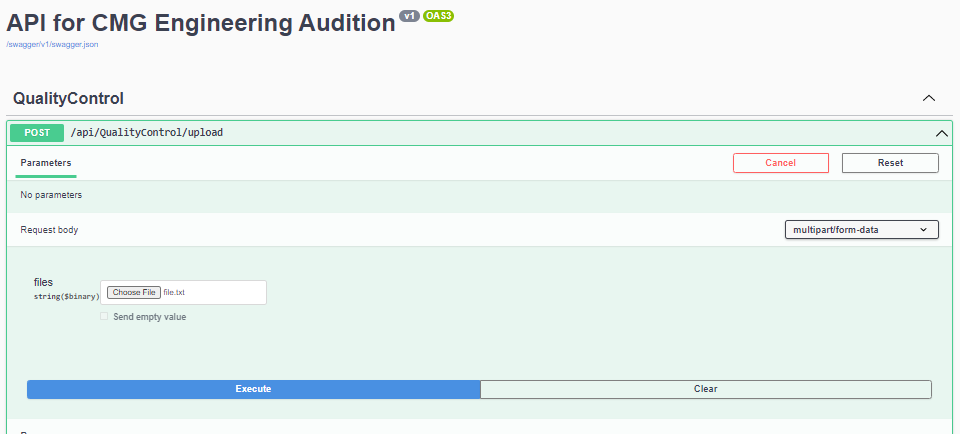
Request on Postman



Response:



Request on Swagger



Response:

