ASP.NET CORE - SOLID AND CLEAN ARCHITECTURE

HELLO!

I am Trevoir Williams

Software Engineer | Lecturer

Expected Knowledge Gained

- Implement SOLID Principles
- Clean Architecture with ASP.NET Core
- Advanced Tools - **MediatR**, **Automapper**, Fluent API and Validation
- Global **Exception Handling** and **Logging**
- Use **Swagger** , **NSwag** and **NSwag** Studio for API integrations
- Implement **CQRS** Pattern
- Build Secure Application
- **Unit** and **Integration testing**
- How to cleanly integrate **third-party services**
- Application **Deployment (Azure** and **IIS)**

•

Course Requirements

- Visual Studio 2019 and .NET 5 (or Latest Version)
- C#/.NET Programming Knowledge
- Some Database Knowledge

SEE YOU SOON!

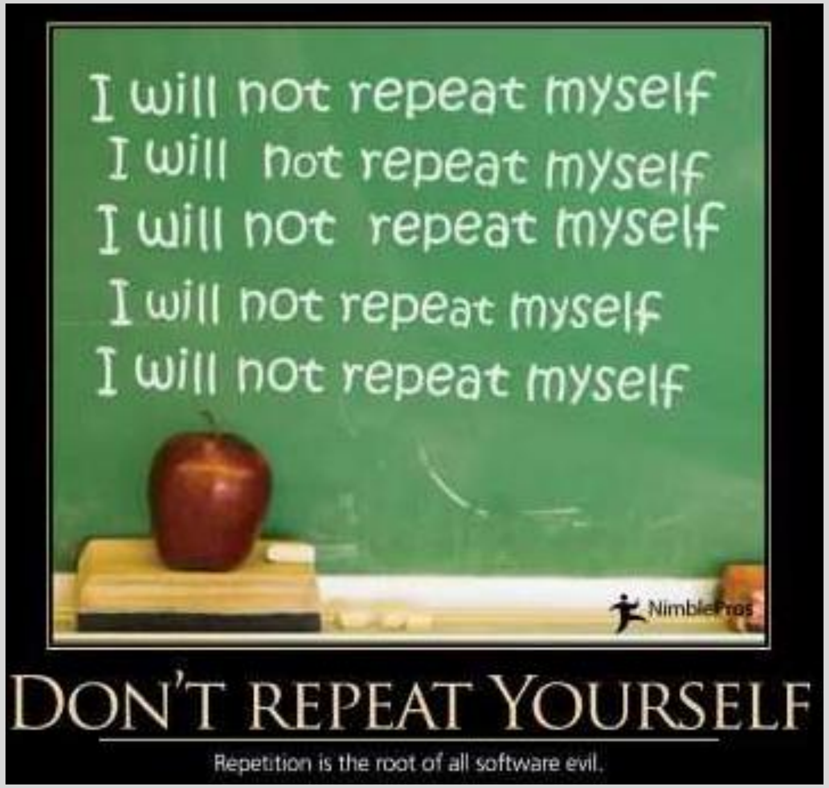
Separation Of Concerns & Single Responsibility

- S in SOLID
- Foundation of Object Oriented Programming
- Each class or block of code should do ONE THING.
- Can be extended to layers, where each layer is in-charge of one thing.
- Concept of splitting functionality into blocks
  - Each addressing a specific concern
  - One block of code shouldn’t be trying to do many different things
- Promotes Modularity
  - Each Module encapsulates all logic for the specific feature set.
  - E.g. Build a Component for Logging, Different from Emailing, etc.

•

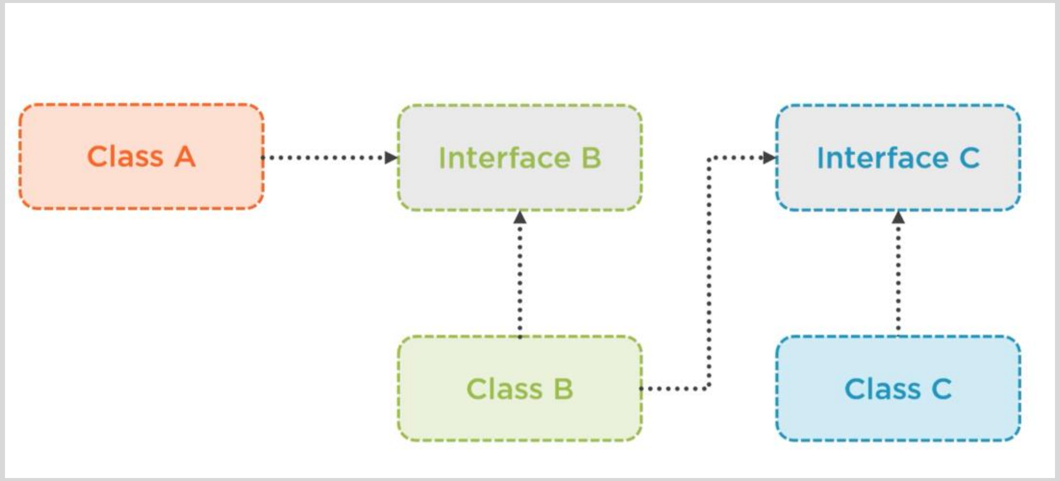
DRY – Don’t Repeat Yourself

- Less Code repetition
    - One implementation point for code in your application.
    - Easier to maintain and make changes.
    - The Single Responsibility Principle relies on DRY.
    - The **Open/Closed Principle** (O in SOLID) only works when DRY is followed.
        - **We should strive to write code that doesn’t have to be changed every time the requirements change**.



Dependency Inversion

- The D in SOLID
- Promotes Loose-Coupling in applications
- Dependencies should point to abstractions,
  - Allows for easier maintenance and modifications to function logic
  - Reduces direct dependencies between classes
- Allows for easier code sharing between dependent classes.



Understanding Clean Architecture

All-In-One Architecture

Pros:

- Easier to deliver
- Can be stable and a long term solution

Cons:

- Hard to Enforce SOLID Principles
- Harder to maintain as project grows
- Harder To Test

Understanding Clean Architecture

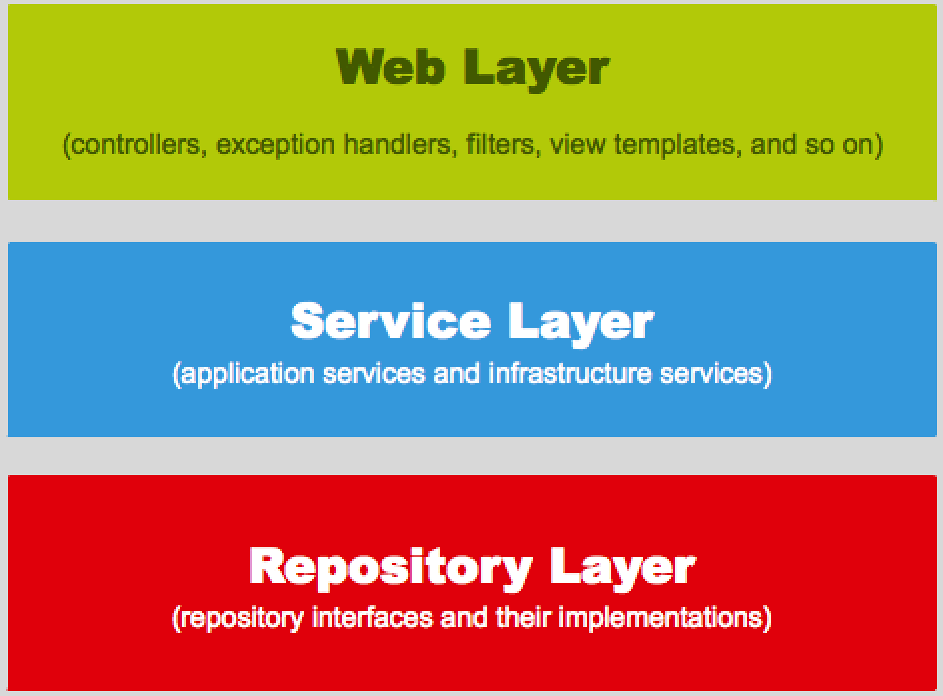
Layered Architecture

Pros:

- Better enforcing od SOLID principles
- Easier to maintain larger code base

Cons:

- Layers are dependent
- Still acts as one application
- Logic is sometimes scattered across layers

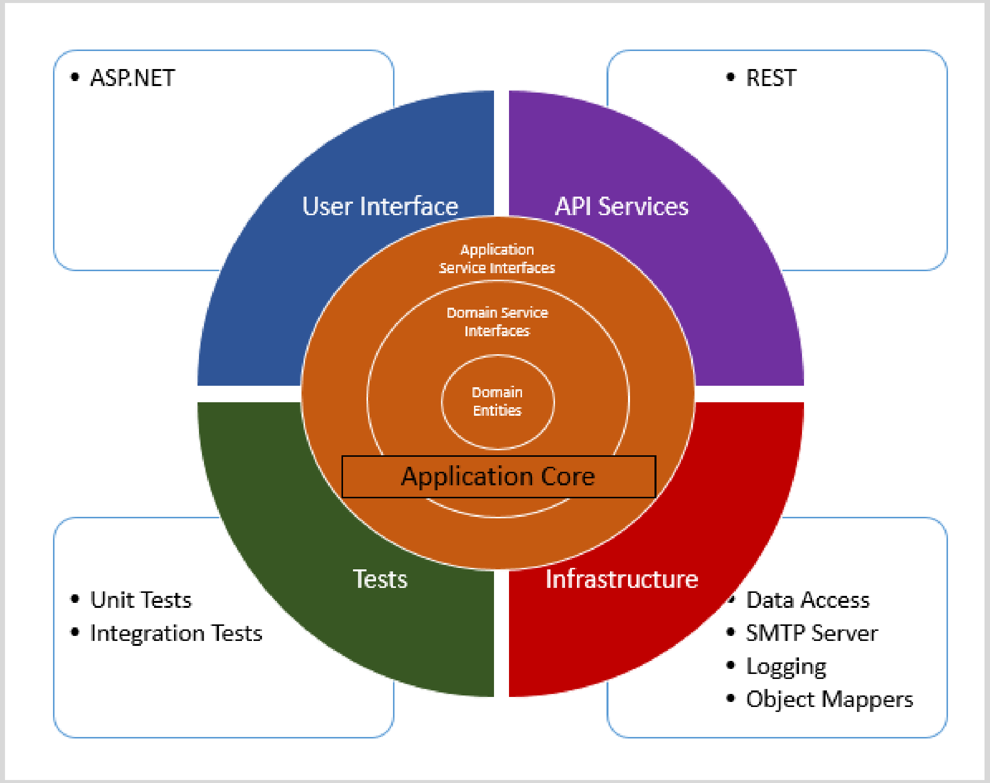

Web Layer
(controllers, exception handlers, filters, view templates, and so on)
Service Layer
(application services and infrastructure services)
Repository Layer
(repository interfaces and their implementations)

Credit: Martin Ledvinka

Understanding Clean Architecture

Onion Architecture

- **Pros:**
  - It provides better testability as unit tests can be created for separate layers
  - Easier to make changes in code base without directly affecting other modules.
  - Promotes loose coupling
- **Cons:**
  - Learning Curve
  - Time Consuming



Credit: Lori Peterson

Understanding Clean Architecture

- **Be Careful! Not every application needs ‘Clean Architecture’**

- Do you **REALLY** need it?
  - Good Software meets the business needs
  - Maintainable software increases the lifespan of the software.

- What is the scale of the application?
  - Not every project needs ‘Clean Architecture’ from day one.
  - Start small and extend as needed.