



# Clean Architecture in ASP.NET Core 6

---

STEVE SMITH

NIMBLEPROS.COM | @ARDALIS | [STEVE@NIMBLEPROS.COM](mailto:STEVE@NIMBLEPROS.COM)

MENTOR | TRAINER | COACH | FORCE MULTIPLIER

# Learn More After Today

---

## 1) Pluralsight

- Domain-Driven Design Fundamentals  
<https://www.pluralsight.com/courses/domain-driven-design-fundamentals>

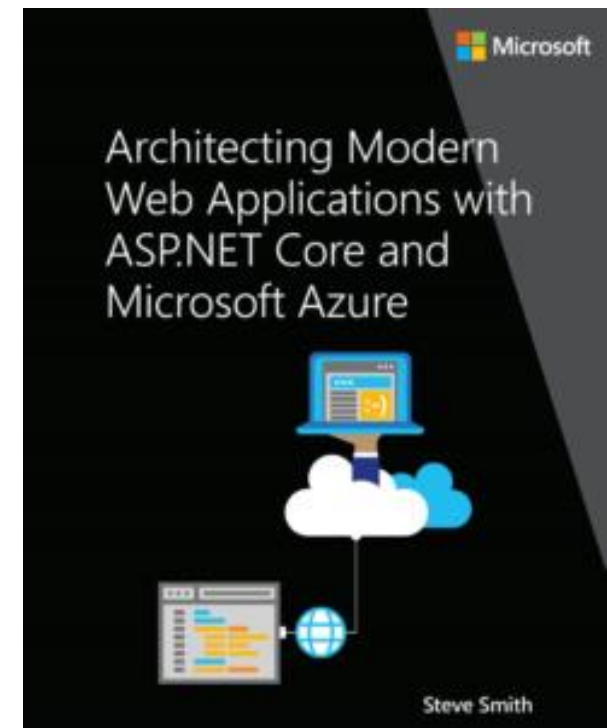
## 2) Microsoft FREE eBook/Sample App (updated for .NET Core 3.1)

- eShopOnWeb eCommerce Sample <https://ardalis.com/architecture-ebook>

## 3) Contact me for mentoring/training for your company/team

- <https://nimblepros.com>

## 4) Developer Career Mentoring at [devBetter.com](https://devbetter.com)



# Weekly Dev Tips

Podcast and Newsletter (and stream)

---

- [Ardalis.com/tips](https://ardalis.com/tips)
- [WeeklyDevTips.com](https://WeeklyDevTips.com)
- Streaming at [twitch.tv/ardalis](https://twitch.tv/ardalis) Fridays



**WEEKLY DEV TIPS**  
WITH STEVE SMITH (@ardalis)

# Questions

---

HOPEFULLY YOU'LL KNOW THE ANSWERS WHEN WE'RE DONE



Why do we **separate** applications into multiple projects?

What are some principles we can apply when organizing our software modules?

How does the **organization** of our application's  
**solution** impact **coupling**?

What **problems** result from certain common approaches?



How does Clean Architecture address these problems?

# Principles

---

A BIT OF GUIDANCE





# SEPARATION OF CONCERNS

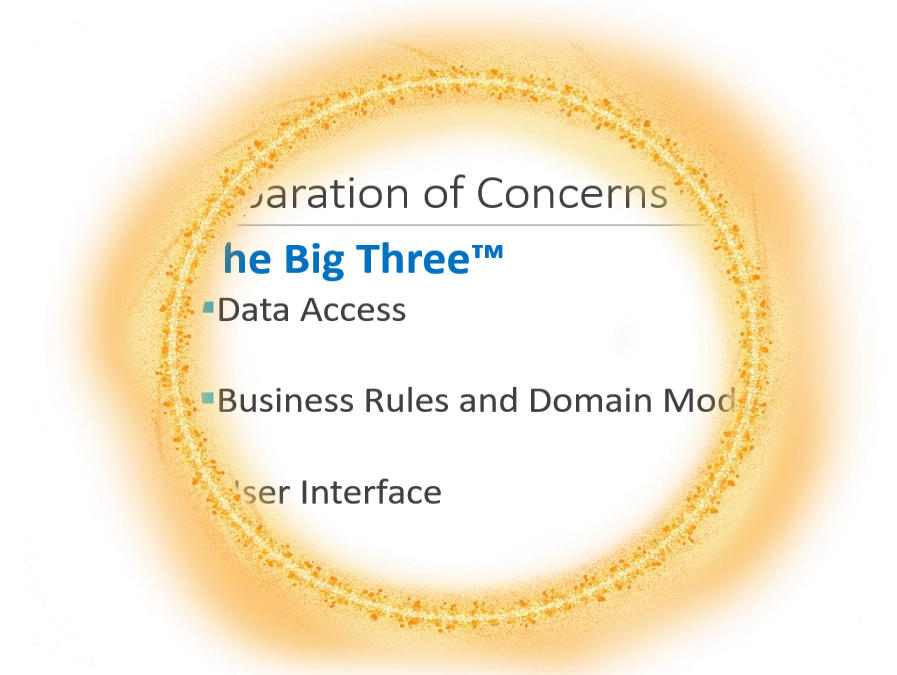
Don't let your plumbing code pollute your software.



# Separation of Concerns

---

Avoid mixing different code responsibilities in the same (method | class | project)



# Separation of Concerns

---

## **The Big Three™**

- Data Access
- Business Rules and Domain Model
- User Interface



# SINGLE RESPONSIBILITY

Avoid tightly coupling your tools together.

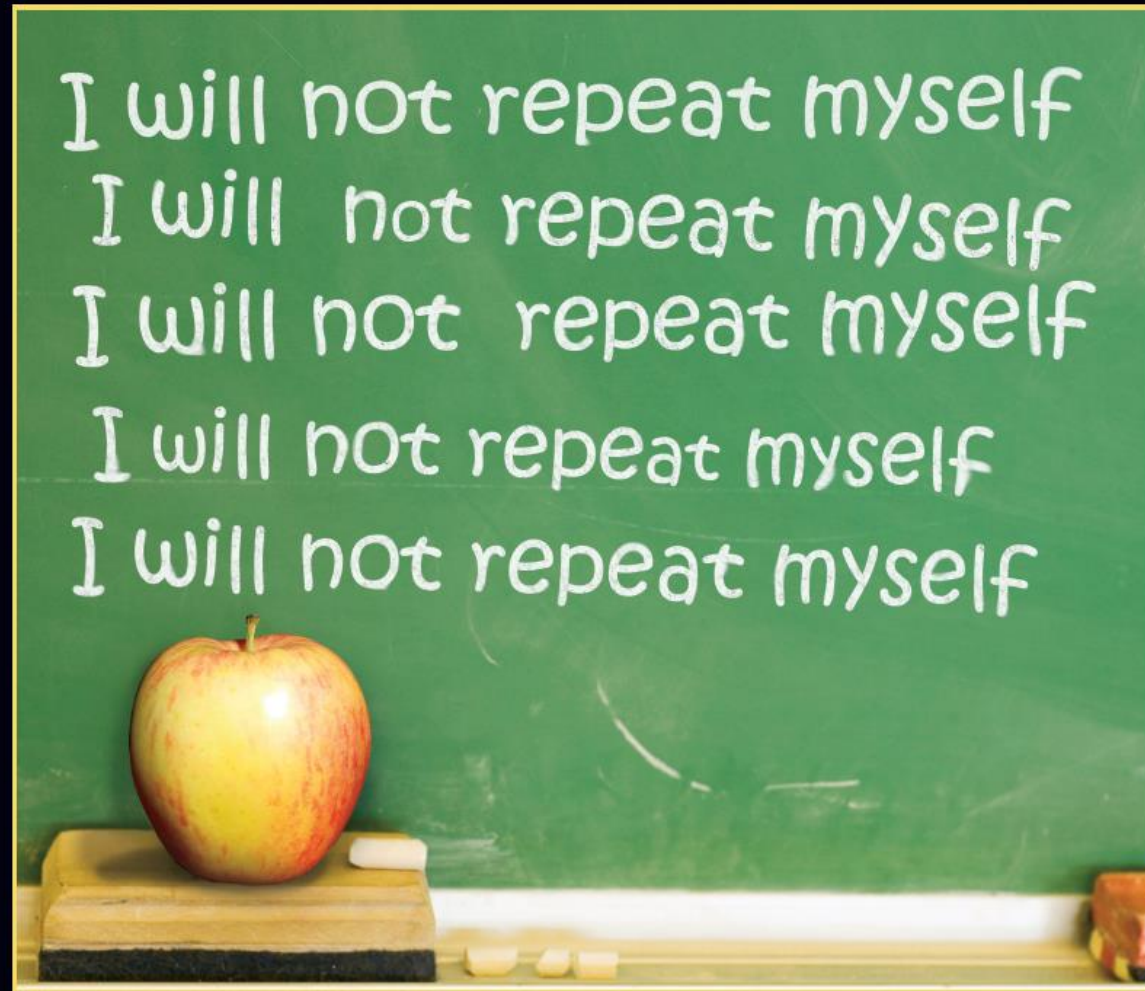
Clean Architecture to ASP.NET Core 6 | @ardalis

# Single Responsibility

---

Works in tandem with **Separation of Concerns**

Classes should focus on a **single responsibility** – a single **reason to change**.



# DON'T REPEAT YOURSELF

Repetition is the root of all software evil.



# Following Don't Repeat Yourself...

---

- Refactor *repetitive code* into **functions**
- Group functions into **cohesive classes**
- Group classes into **folders and namespaces** by
  - Responsibility
  - Level of abstraction
  - Etc.
- Further group class folders into **projects**

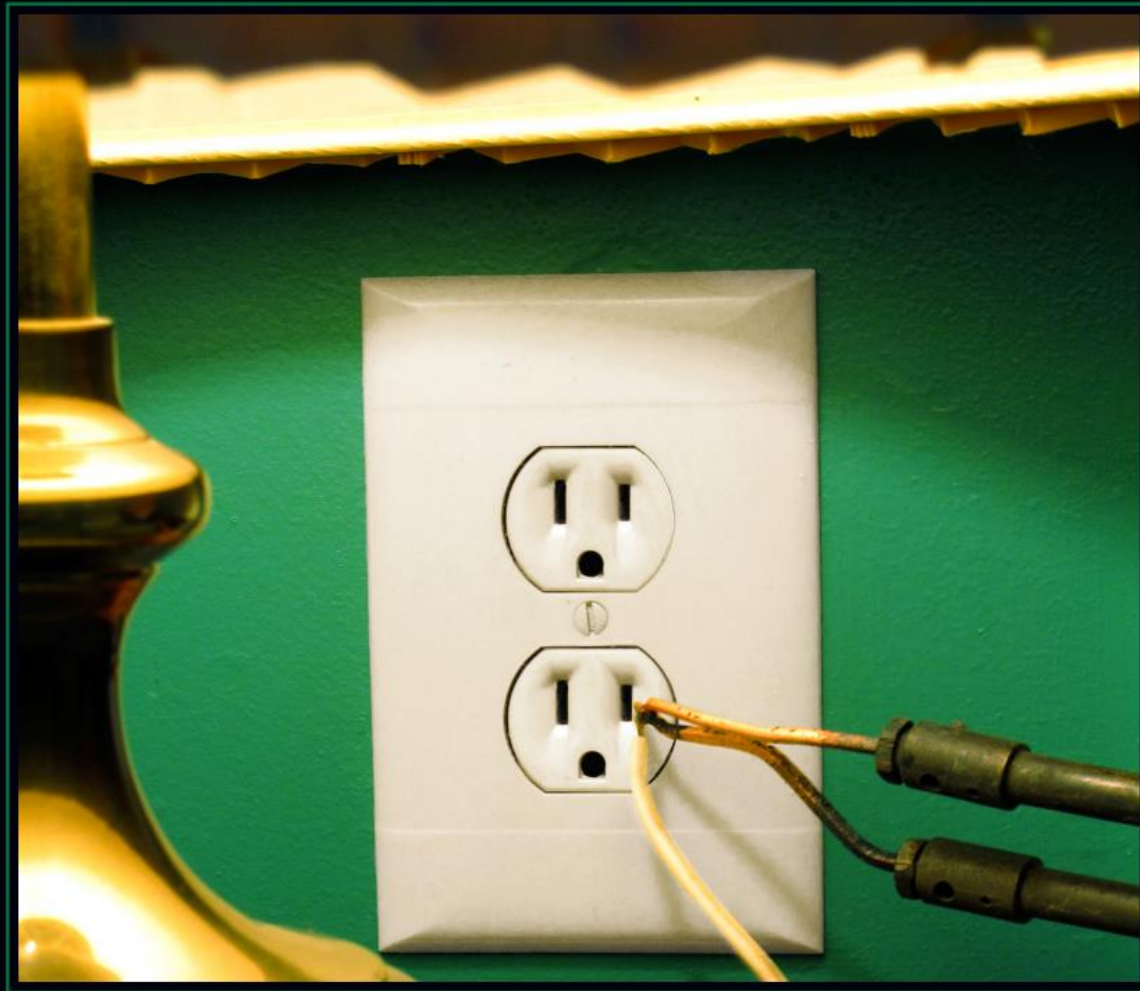
# More Grouping Options to Explore

---

Aggregates

Features

Bounded Contexts



# DEPENDENCY INVERSION

Would you solder a lamp directly to the electrical wiring in a wall?

# Invert (and inject) Dependencies

---

Both high level classes and implementation-detail classes should depend on **abstractions (interfaces)**.

# Invert (and inject) Dependencies

---

Classes should follow **Explicit Dependencies Principle**:

- Request **all** dependencies via their constructor
- Make your types honest, not deceptive



# Invert (and inject) Dependencies

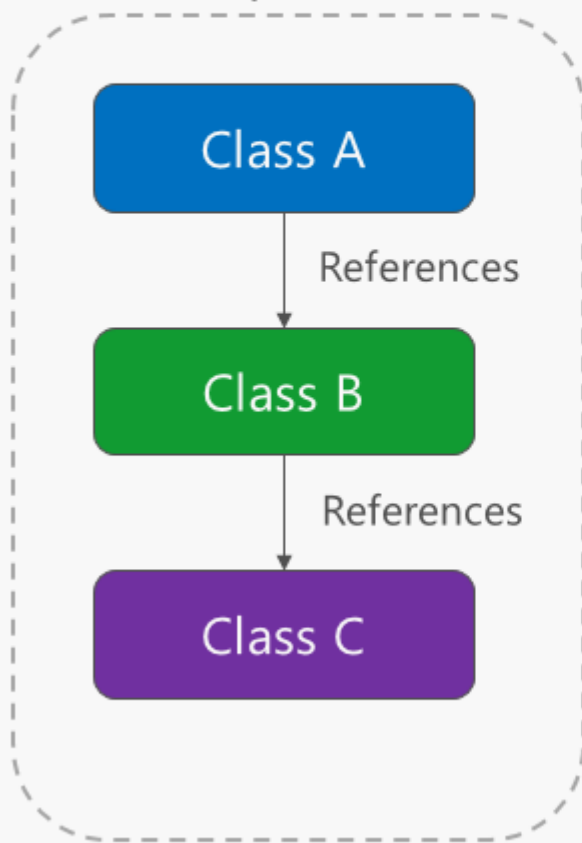
---

Corollary: **Abstractions/interfaces must be defined somewhere accessible by:**

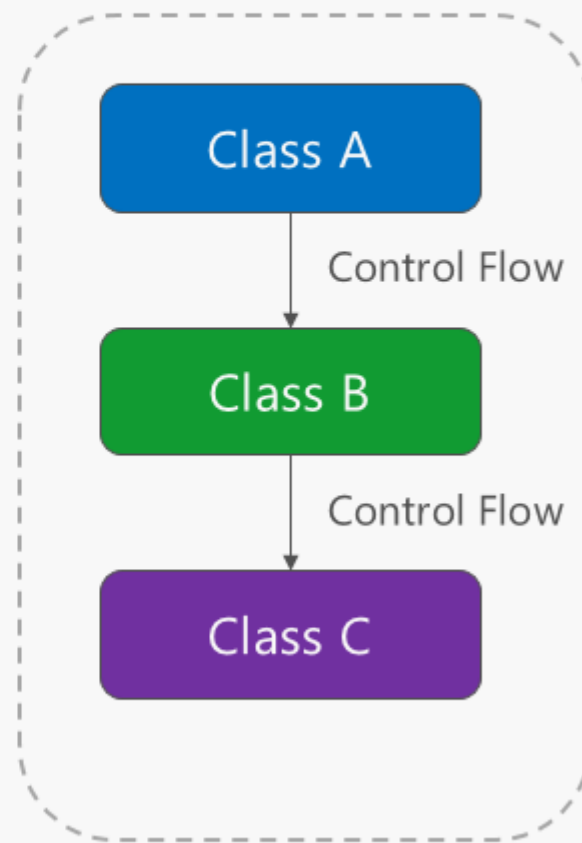
- Low level implementation services
- High level business services
- User interface entry points

# Direct Dependency Graph

Compile Time

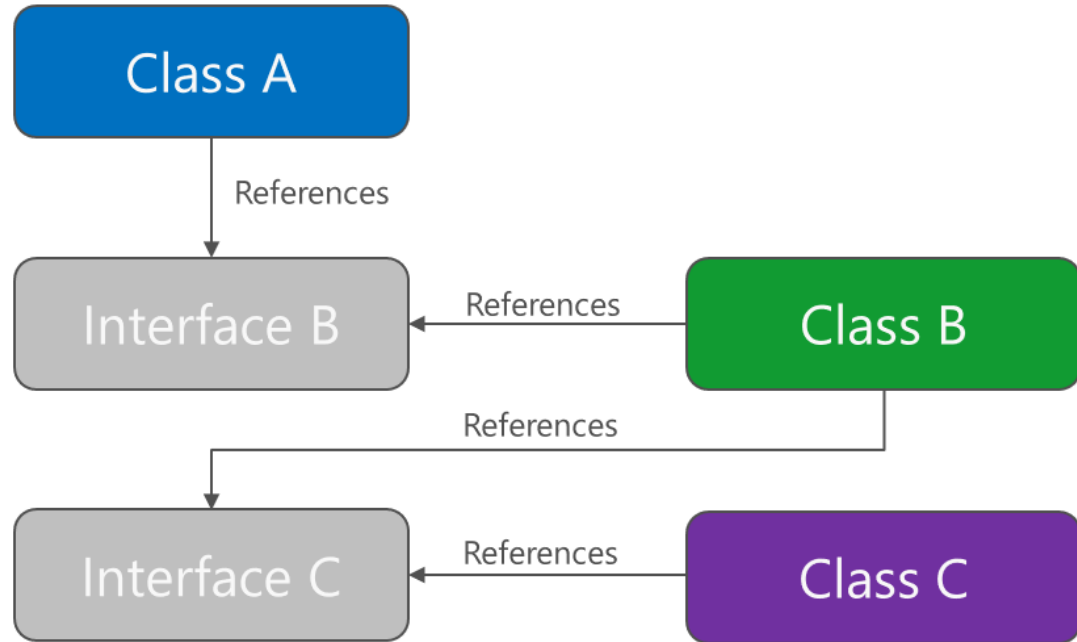


Run Time

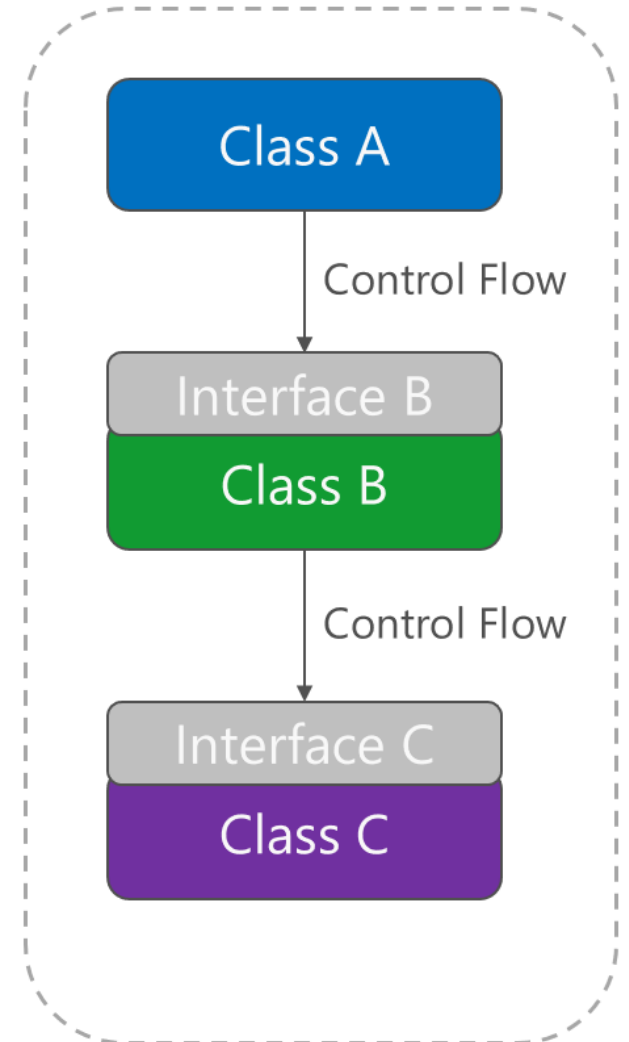


# Inverted Dependency Graph

Compile Time



Run Time





Make the right thing easy  
and the wrong thing hard

---

FORCE DEVELOPERS INTO A “PIT OF SUCCESS”

Make the **right thing easy** and the **wrong thing hard**.

---

UI classes shouldn't depend directly on infrastructure classes

- How can we **structure our solution** to help enforce this?

Make the **right thing easy** and the **wrong thing hard**.

---

Business/domain classes shouldn't depend on infrastructure classes

- How can our **solution design** help?

Make the **right thing easy** and the **wrong thing hard**.

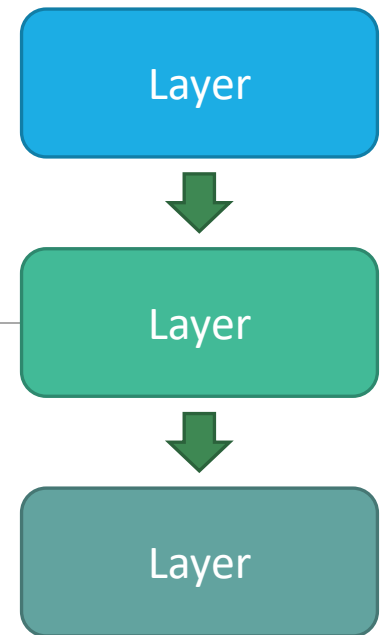
---

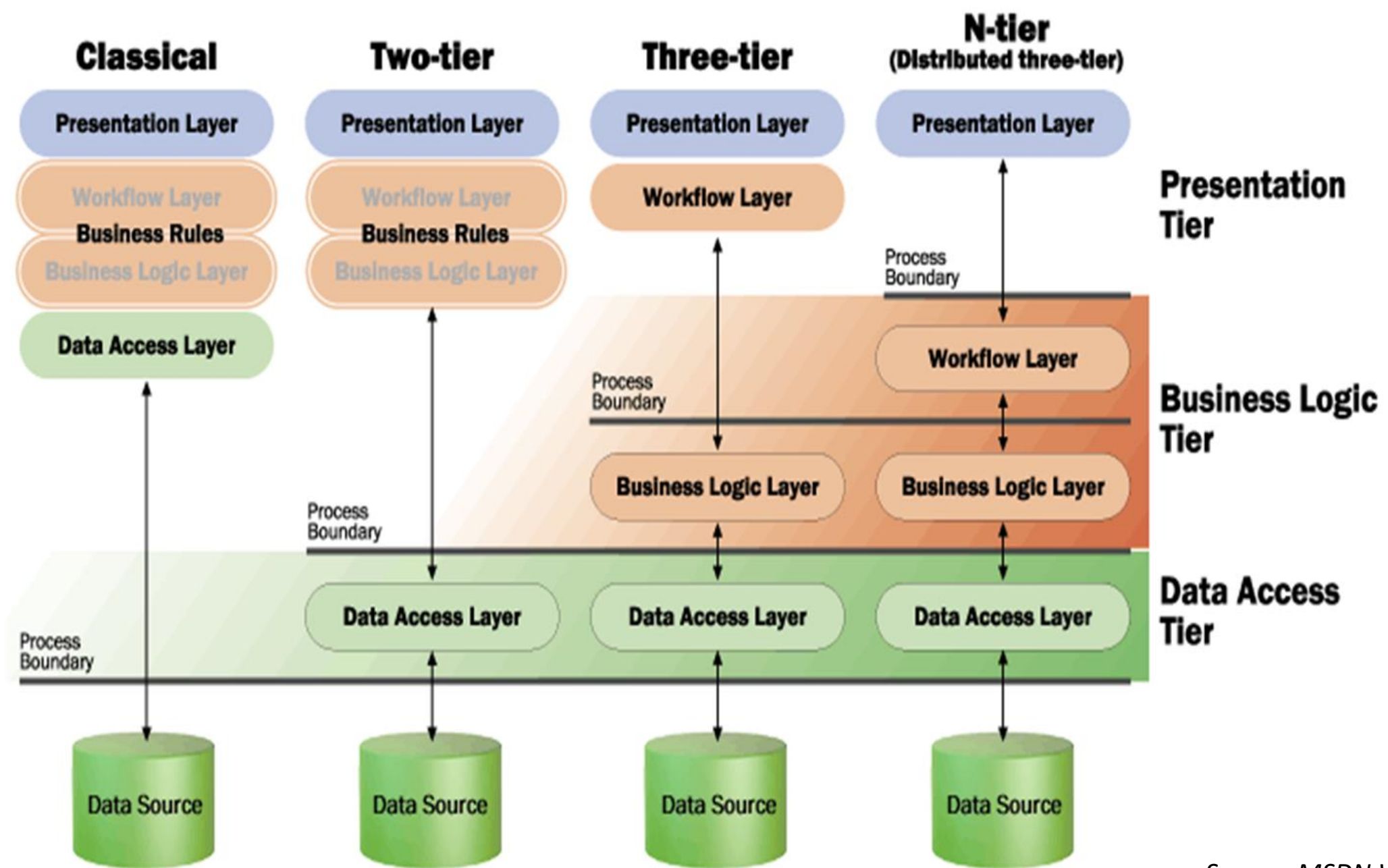
**Repetition of** (query logic, validation logic, policies, error handling, anything) **is a problem...**

- What **patterns** can we apply to make avoiding repetition **easier than copy/pasting**?

# “Classic” N-Tier Architecture

OR N-LAYER

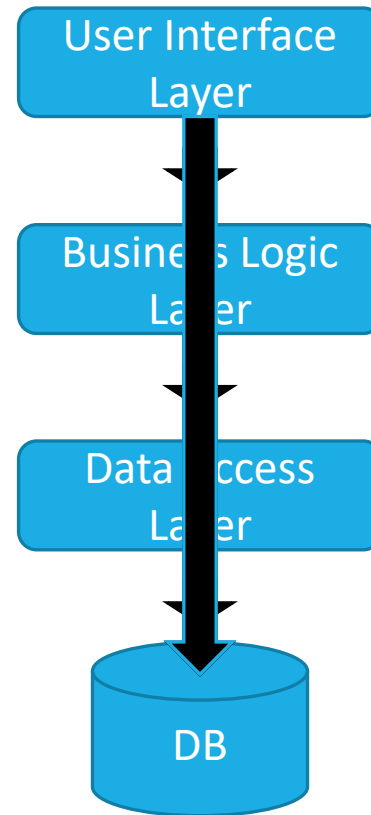




Source: MSDN Website, 2001

# Transitive Dependencies

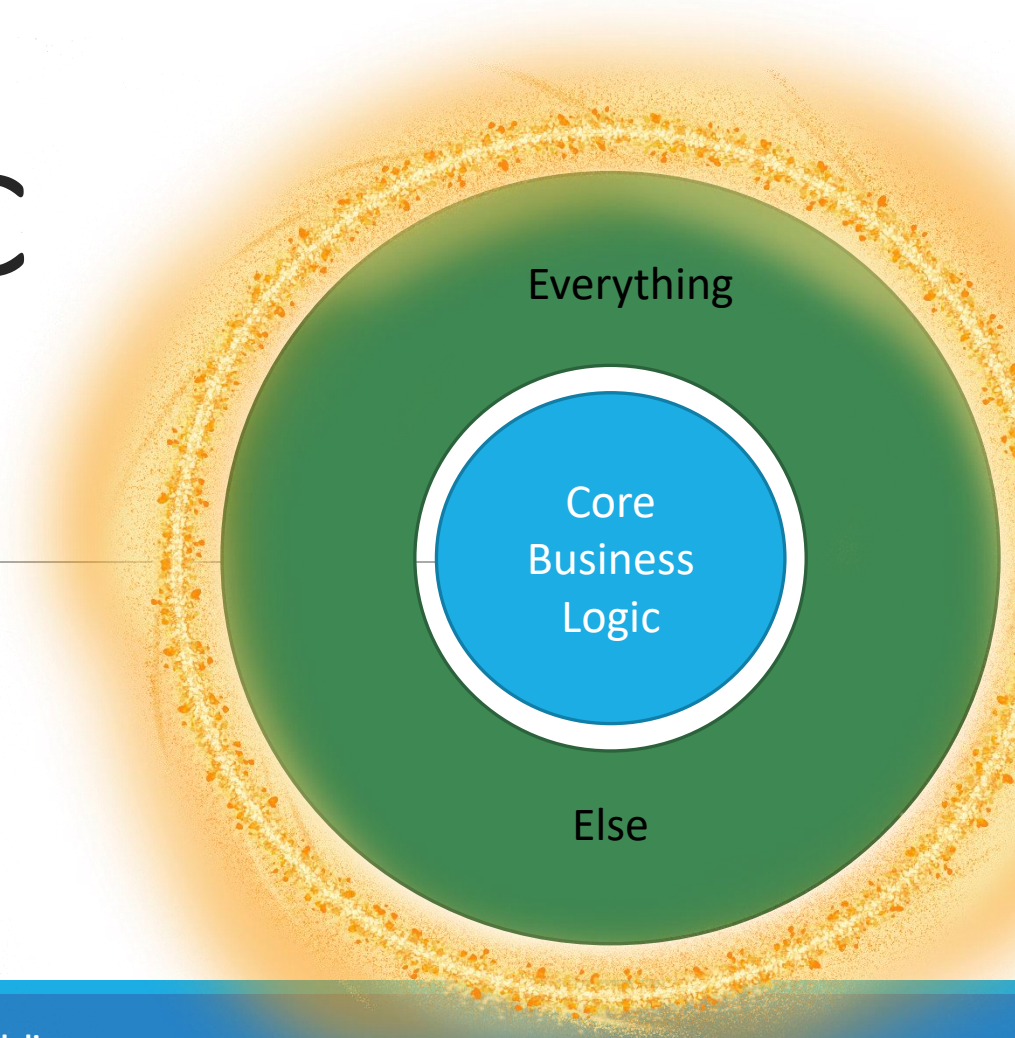
---



**Everything**  
Depends on the *database*

# Domain-Centric Design

AND THE CLEAN ARCHITECTURE





# Domain Model

---

Not just business logic, but also:

A **model** of the problem space composed of Entities, Interfaces, Services, and more.

**Interfaces** define contracts for working with domain objects

**Everything** in the application (including infrastructure and data access) depends on these interfaces and domain objects

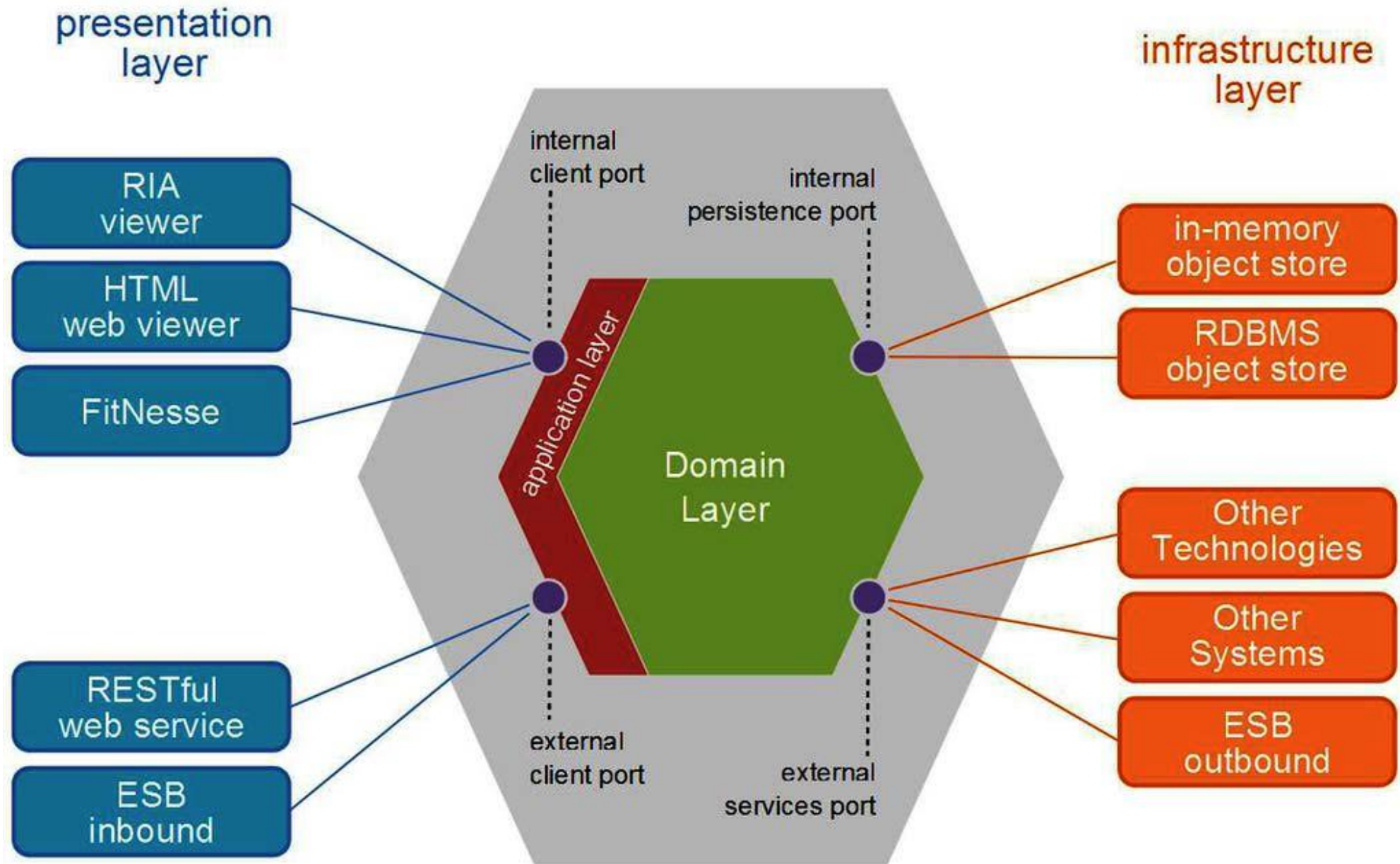
# Clean Architecture

---

## Onion Architecture

## Hexagonal Architecture

## Ports and Adapters



# Clean Architecture “Rules”

---

The Application **Core** contains the **Domain Model**

# Clean Architecture “Rules”

---

All projects depend on the Core project;  
**dependencies point inward** toward this core

# Clean Architecture “Rules”

---

Inner projects define **interfaces**;

Outer projects **implement** them

# Clean Architecture “Rules”

---

**Avoid direct dependency** on the Infrastructure project  
(except from **Integration Tests** and possibly **Startup.cs**)

# Clean Architecture Features

---

## Framework Independent

- You can use this architecture with ASP.NET (Core), Java, Python, etc.
- It doesn't rely on any software library or proprietary codebase.



# Clean Architecture Features

---

## Database Independent

- The vast majority of the code has no knowledge of persistence details.
- This knowledge may exist in just one class, in one project that no other project references.

# Clean Architecture Features

---

## UI Independent

- Only the UI project cares about the UI.
- The rest of the system is UI-agnostic.

# Clean Architecture Features

---

## Testable

- Apps built using this approach, and especially the core domain model and its business rules, are easy to test.

# Refactoring to a Clean Architecture

---

**Best** to start from a properly organized solution

- See <https://github.com/ardalis/CleanArchitecture>

Next-best: Start from an application consisting of just a single project

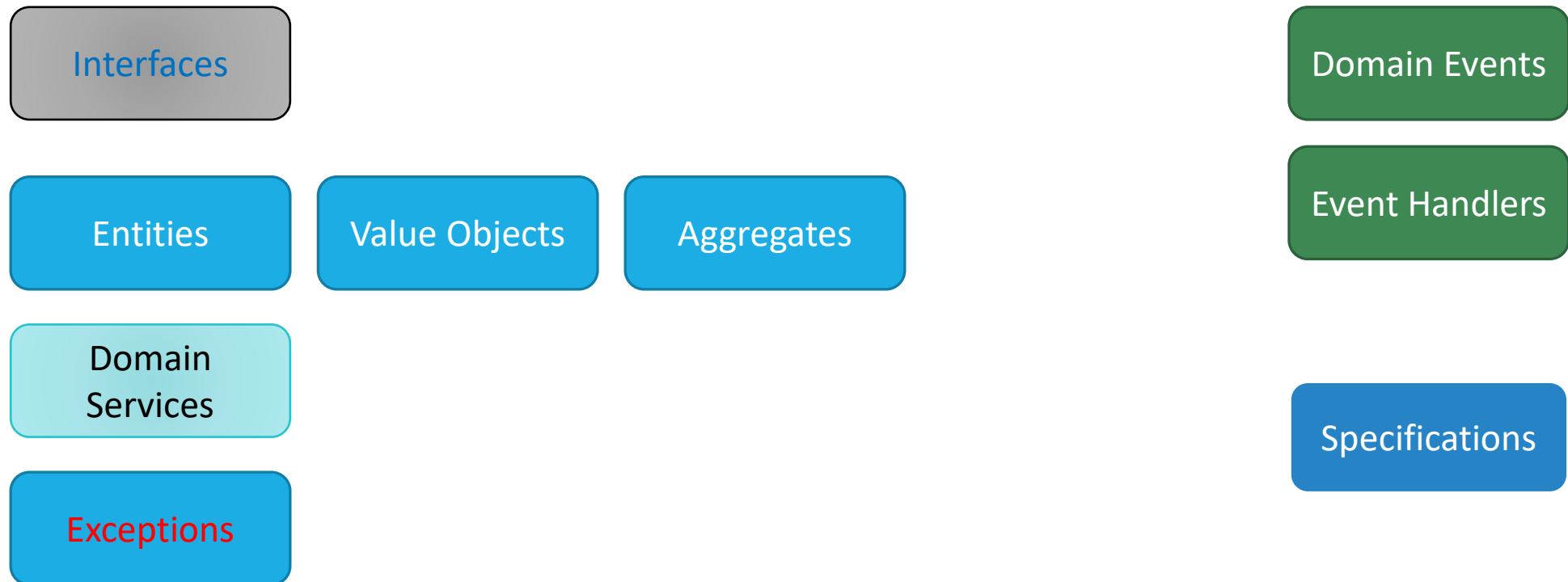
**Most difficult:** Large, existing investment in multi-layer architecture without abstractions or DI

# The Core Project (domain model)

---

Minimal dependencies – none on *Infrastructure*.

What Goes in Core:

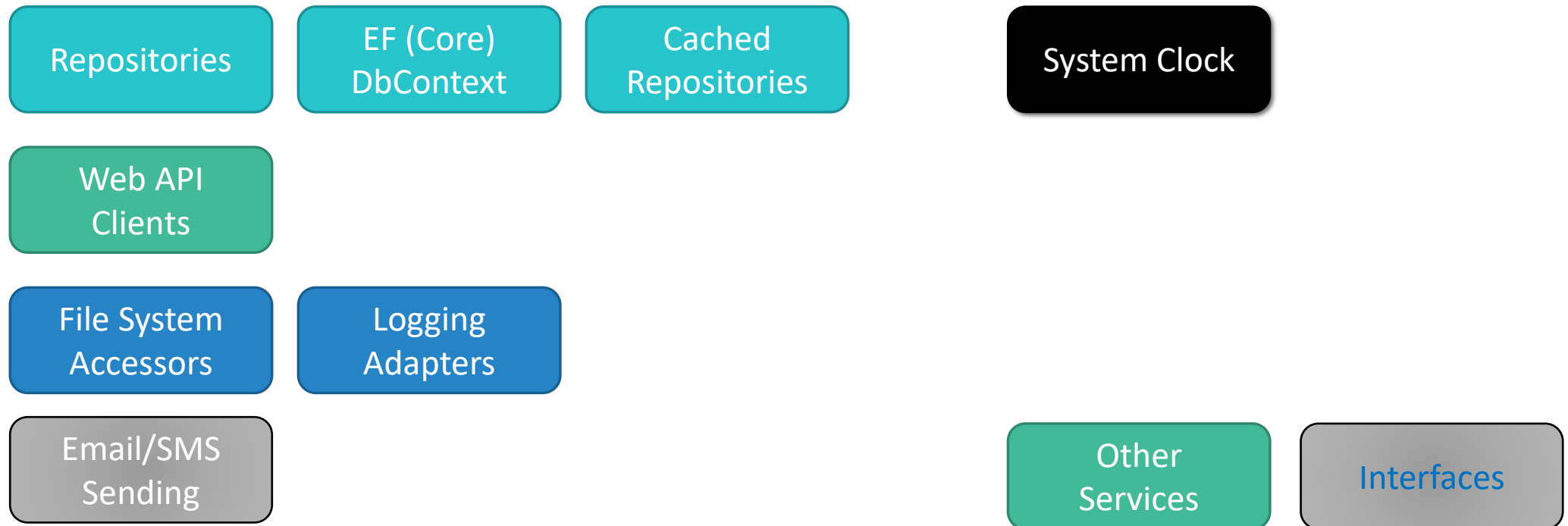


# The Infrastructure Project

---

All dependencies on out-of-process resources.

What Goes in Infrastructure:

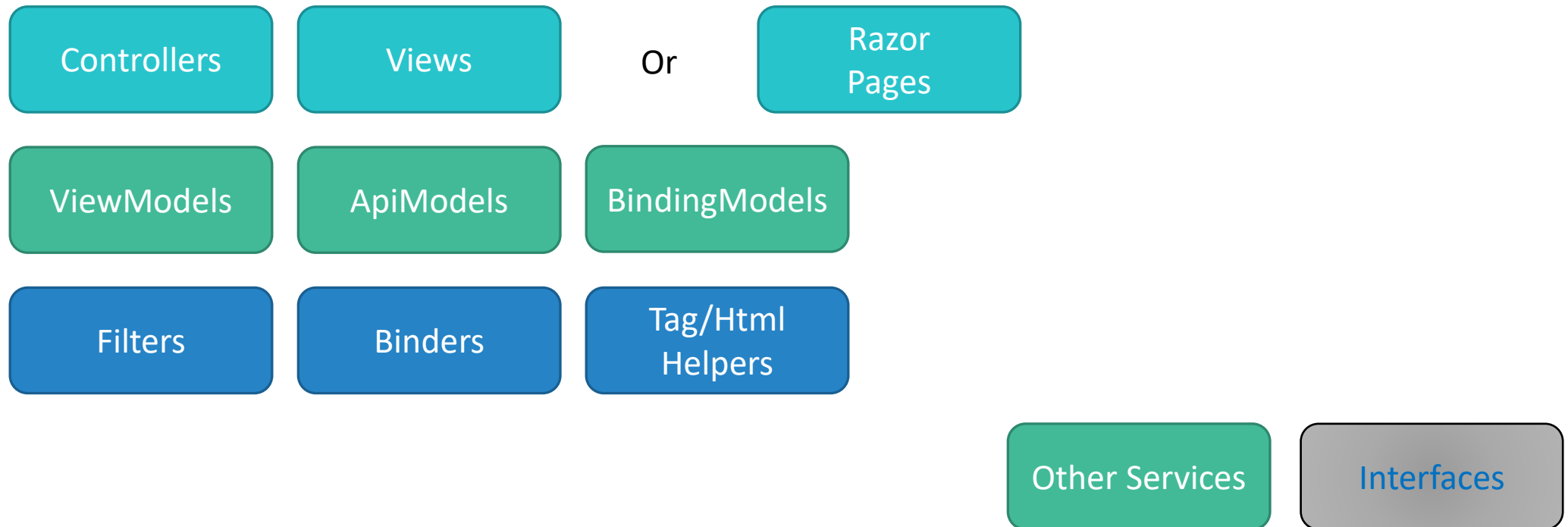


# The Web Project

---

All dependencies on out-of-process resources.

What Goes in Web:



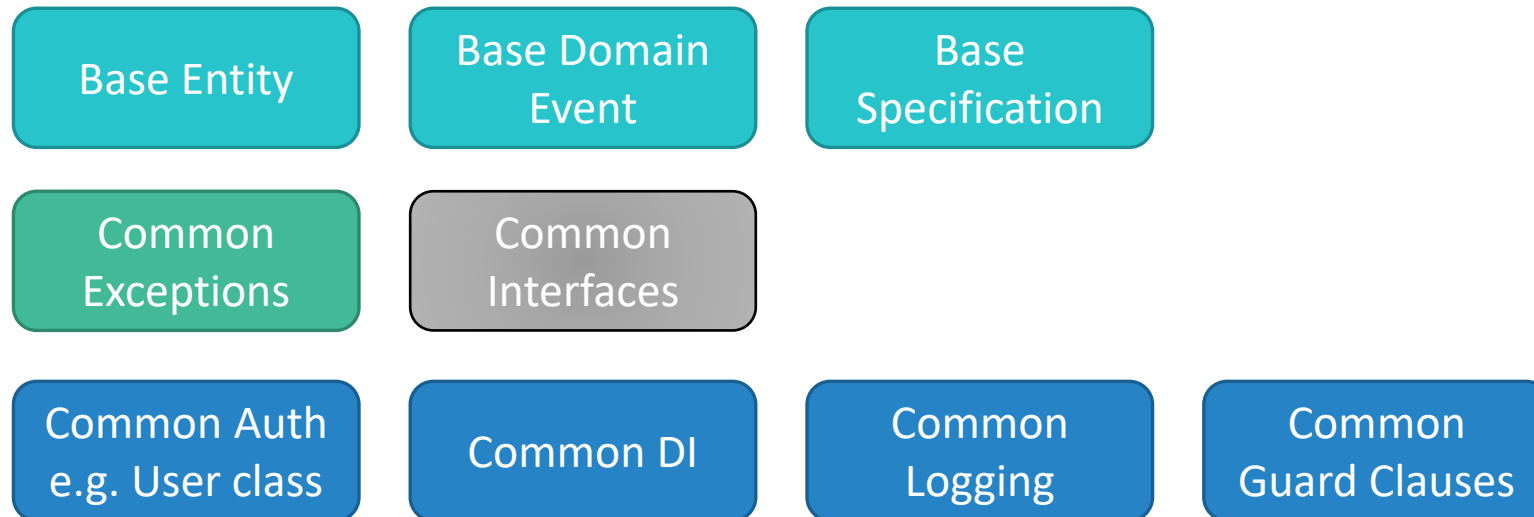
# Sharing Between Solutions: Shared Kernel

---

Common Types May Be Shared Between Solutions. Will be referenced by **Core** project(s).

Ideally distributed as **Nuget Packages**.

What Goes in Shared Kernel:





# Guard Clauses?

---

BAD EXAMPLE

```
public void ProcessOrder(Order order, Custom customer)
{
    if(order != null)
    {
        if(customer != null)
        {
            // process order here
        } else {
            throw new ArgumentNullException(nameof(customer), customer);
        }
    } else {
        throw new ArgumentNullException(nameof(order), order);
    }
}
```

# *Guard Clauses?*

---

```
public void ProcessOrder(Order order, Customer customer)
{
    if(order == null) throw new ArgumentNullException(nameof(order),
order);

    if(customer==null) throw new ArgumentNullException(nameof(customer),
customer);

    // process order here
}
```

# Guard Clauses?

---

Simple checks for input that use common rules and exceptions.

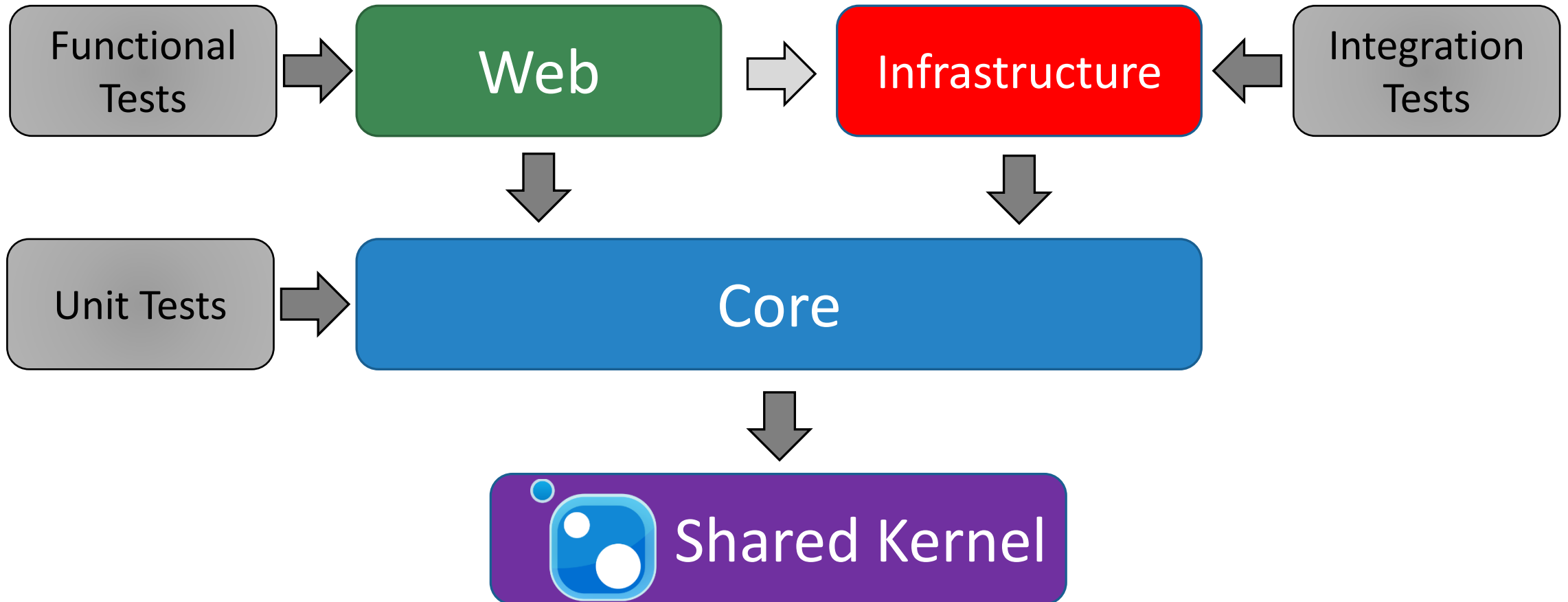
Nuget Package: [Ardalis.GuardClauses](https://github.com/ardalis/GuardClauses) (<https://github.com/ardalis/GuardClauses>)

**Example:**

```
public void ProcessOrder(Order order, Customer customer)
{
    Guard.Against.Null(order, nameof(order));
    Guard.Against.Null(customer, nameof(customer));
    // process order here
}
```

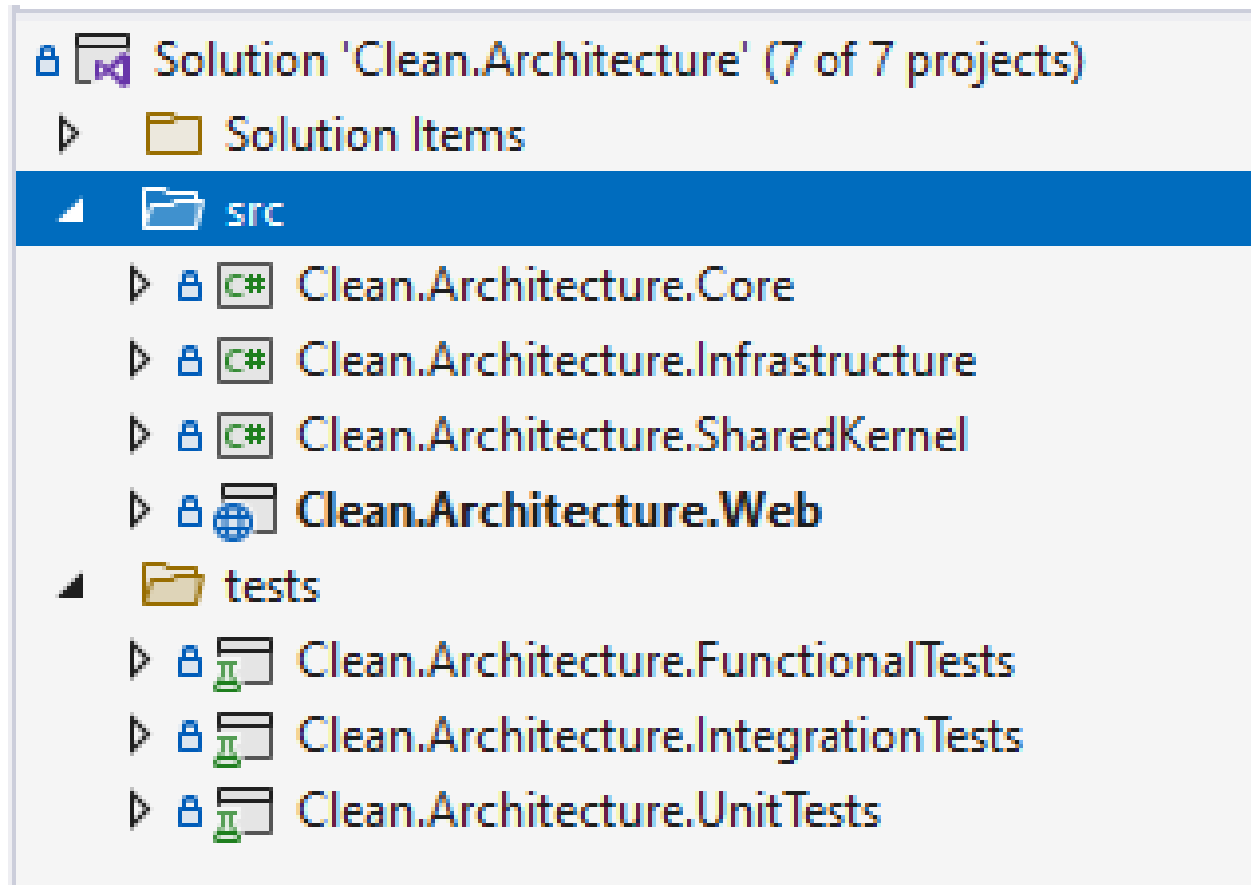
# Solution Structure – Clean Architecture

---



# Typical (Basic) Folder Structure

---



# Walkthroughs

---

LET'S SEE SOME EXAMPLES

# eShopOnWeb

The screenshot displays the GitHub repository page for `dotnet-architecture / eShopOnWeb`. The repository is public and has 485 watchers, 3.7k forks, and 7.4k stars. The main navigation bar includes links to Code, Issues (4), Pull requests (4), Actions, Projects, Wiki, Security, Insights, and Settings. The repository is currently on the `main` branch, with 13 branches and 4 tags.

The file list shows the following files and their commit history:

File	Commit Message	Commit Time
<code>.devcontainer</code>	Update Dockerfile and launch.json to 5.0 (#494)	16 months ago
<code>.github</code>	Shady nagy/net6 (#614)	5 months ago
<code>.vscode</code>	Update Dockerfile and launch.json to 5.0 (#494)	16 months ago
<code>src</code>	Bump System.IdentityModel.Tokens.Jwt from 6.16.0 to 6.17.0 (#699)	4 days ago
<code>tests</code>	Fix docker environment issues (#694)	4 days ago
<code>.dockerignore</code>	Added docker support	4 years ago
<code>.editorconfig</code>	net6conversion (#642)	4 months ago
<code>.gitignore</code>	Fix for #375 to add devcontainer/extensions files	2 years ago
<code>CodeCoverage.runsettings</code>	Test cleanup and adding code coverage	3 years ago
<code>LICENSE</code>	Create LICENSE	5 years ago
<code>README.md</code>	Feature/migrate to minimal api (#662)	3 months ago
<code>docker-compose.dcpoj</code>	Adding docker-compose project (#302)	3 years ago
<code>docker-compose.override.yml</code>	Updating Blazor Admin (#442)	2 years ago

The sidebar on the right contains the following information:

- About:** Sample ASP.NET Core 6.0 reference application, powered by Microsoft, demonstrating a layered application architecture with monolithic deployment model. Download the eBook PDF from docs folder.
- Tags:** [docs.microsoft.com/en-us/dotnet/standa...](#)
- Tags:** [microsoft](#), [ddd](#), [dotnetcore](#), [rest-api](#), [clean-code](#), [architecture](#), [ebook](#), [design-patterns](#), [clean-architecture](#), [efcore](#), [asp-net-core](#), [asp-net-core-mvc](#), [monolith](#), [ddd-patterns](#), [software-architecture](#), [ddd-architecture](#), [ddd-sample](#)
- Readme:** [Readme](#)
- MIT License:** [MIT License](#)
- Stars:** 7.4k stars
- Watchers:** 485 watching
- Forks:** 3.7k forks

# Ardalis.CleanArchitecture

The screenshot displays the GitHub repository page for **ardalis / CleanArchitecture**, which is a public template. The repository has 14 issues, 14 pull requests, 25 branches, and 5 tags. It is being watched by 412 people, has 1.8k forks, and 10k stars.

The main content area shows a list of files and their commit history:

File	Commit Message	Commit Date
.github	Update publish.yml	5 months ago
.template.config	update naming	5 months ago
src	Updating packages (#328)	last month
tests	Updating packages (#328)	last month
.editorconfig	Update private field naming convention	5 months ago
.gitignore	Adding Serilog, AppInsights Sink, and updated existing log messages t...	2 months ago
CleanArchitecture.sln	Update solution items	5 months ago
CleanArchitecture.nuspec	Update CleanArchitecture.nuspec	13 days ago
LICENSE	Initial commit	6 years ago
README.md	Update README.md	5 months ago
global.json	add global json (#288)	4 months ago

The right sidebar contains the **About** section, which describes the repository as a "Clean Architecture Solution Template: A starting point for Clean Architecture with ASP.NET Core". It also lists the repository's tags: **dotnet**, **ddd**, **architecture**, and **clean-architecture**. Below this, the **Releases** section shows 5 tags and a link to "Create a new release".



# Remember

---

Manage coupling and dependencies

Enforce coupling using project structure and architecture decisions

Learn more:

- <https://github.com/dotnet-architecture/eShopOnWeb>
- <https://github.com/ardalis/cleanarchitecture>

Get Help:

- <https://nimblepros.com/>



# What belongs in actions/handlers?

---

Controller Actions (or Page Handlers) should:

- 1) Accept task-specific types (ViewModel, ApiModel, BindingModel)
- 2) Perform and handle model validation (ideally w/filters)
- 3) “Do Work” (*More on this in a moment*)
- 4) Create any model type required for response (ViewModel, ApiModel, etc.)
- 5) Return an appropriate Result type (View, Page, Ok, NotFound, etc.)

# “Do Work” – Option One

---

## Repositories and Entities

- 1) Get entity from an injected Repository
- 2) Work with the entity and its methods.
- 3) Update the entity's state using the Repository

Great for simple operations

Great for CRUD work

Requires [mapping](#) between web models and domain model within controller

```
[HttpPost("{itemId}")]
```

0 references | Steve Smith, 12 minutes ago | 1 author, 1 change | 0 requests | 0 exceptions

```
public IActionResult MarkComplete(int itemId)
```

```
{
```

```
    var item = _todoRepository.GetById(itemId);
```

```
    item.MarkComplete();
```

```
    _todoRepository.Update(item);
```

```
    return Ok();
```

```
}
```

# “Do Work” – Option Two

---

Work with an [application service](#).

- 1) Pass ApiModel types to service
- 2) Service internally works with repositories and domain model types.
- 3) Service returns a web model type

Better for more complex operations

Application Service is responsible for mapping between models

Keeps controllers lightweight, and with fewer injected dependencies

```
[HttpPost("{itemId}")]
```

0 references | Steve Smith, 13 minutes ago | 1 author, 1 change | 0 requests | 0 exceptions

```
public IActionResult MarkComplete(int itemId)
{
    _appService.MarkComplete(itemId);

    return Ok();
}
```

# “Do Work” – Option Three

---

Work with **commands** and a tool like **Mediatr**

- 1) Use ApiModel types that represent commands (e.g. RegisterUser)
- 2) Send model-bound instance of command to handler using `_mediator.Send()`

No need to inject separate services to different controllers – Mediatr becomes only dependency.

# Instantiate Appropriate Command

---

```
[HttpPost("{itemId}")]
```

0 references | Steve Smith, 14 minutes ago | 1 author, 1 change | 0 requests | 0 exceptions

```
public async Task<IActionResult> MarkComplete(int itemId)
{
    var command = new MarkItemCompleteCommand { Id = itemId };

    await _mediator.Send(command);

    return Ok();
}
```



# Resolve Command w/Model Binding

---

```
[HttpPost("MarkComplete/{Id}")]
```

0 references | 0 changes | 0 authors, 0 changes | 0 requests | 0 exceptions

```
public async Task<IActionResult> MarkComplete(MarkItemCompleteCommand command)
{
    await _mediator.Send(command);

    return Ok();
}
```

# eShopOnWeb

The screenshot shows the GitHub repository page for `dotnet-architecture/eShopOnWeb`. The repository has 466 watches, 5.3k stars, and 2.5k forks. It is currently on the `master` branch, which has 13 branches and 4 tags. The repository is described as a "Sample ASP.NET Core 5.0 reference application, powered by Microsoft, demonstrating a layered application architecture with monolithic deployment model. Download the eBook PDF from docs folder."

The file list shows the following files and their commit history:

File	Description	Commit
<code>.devcontainer</code>	configured devcontainer properly (#384)	7 months ago
<code>.github/workflows</code>	Feature/update net 5 (#487)	13 days ago
<code>.vscode</code>	configured devcontainer properly (#384)	7 months ago
<code>src</code>	Feature/update net 5 (#487)	13 days ago
<code>tests</code>	Feature/update net 5 (#487)	13 days ago
<code>.dockerignore</code>	Added docker support	3 years ago
<code>.gitignore</code>	Fix for #375 to add devcontainer/extensions files	7 months ago
<code>CodeCoverage.runsettings</code>	Test cleanup and adding code coverage	2 years ago
<code>LICENSE</code>	Create LICENSE	3 years ago
<code>README.md</code>	updated version informations in README file	8 days ago
<code>docker-compose.dcproj</code>	Adding docker-compose project (#302)	15 months ago
<code>docker-compose.override.yml</code>	Updating Blazor Admin (#442)	5 months ago
<code>docker-compose.yml</code>	Update docker compose to include new Public API project (#414)	6 months ago
<code>eShopOnWeb.sln</code>	Fix favicon.ico build bug and modify CI to run on PR (#479)	last month

The README.md file shows a build status for `eShopOnWeb Build and Test` as `passing`.

The right sidebar contains the following sections:

- About**: Sample ASP.NET Core 5.0 reference application, powered by Microsoft, demonstrating a layered application architecture with monolithic deployment model. Download the eBook PDF from docs folder.
- docs.microsoft.com/en-us/dotnet/sta...**: Link to the documentation.
- Tags**: `asp-net-core`, `asp-net-core-mvc`, `clean-architecture`, `clean-code`, `ebook`, `microsoft`, `ddd`, `ddd-architecture`, `ddd-patterns`, `ddd-sample`, `software-architecture`, `dotnetcore`, `efcore`, `monolith`, `architecture`, `rest-api`, `design-patterns`.
- Readme**: Link to the README file.
- MIT License**: Link to the MIT License.
- Releases**: 4 tags.
- Packages**: No packages published.

# Ardalis.CleanArchitecture

← → ↺ 🔒 https://github.com/ardalis/cleanarchitecture

Why GitHub? Team Enterprise Explore Marketplace Pricing Search Sign in Sign up

ardalis / CleanArchitecture Template Sponsor Watch 354 Star 5.6k Fork 1.1k

<> Code Issues 5 Pull requests 1 Actions Projects Security Insights

master 10 branches 4 tags Go to file Code

ardalis Update README.md ✓ d10e1f3 6 days ago 162 commits

.github	Update dotnetcore.yml	9 days ago
.template.config	Add dotnet template support (#144)	14 days ago
.vscode	Adding list services nuget package (#85)	13 months ago
src	code cleanup	6 days ago
tests	code cleanup	6 days ago
.gitignore	Update static files	6 days ago
Clean.Architecture.sln	Re-ordered solution to make Web the default startup project	6 days ago
CleanArchitecture.nuspec	update nuspec; add icon	7 days ago
LICENSE	Initial commit	4 years ago
README.md	Update README.md	6 days ago
icon.png	update nuspec; add icon	7 days ago

About  
A starting point for Clean Architecture with ASP.NET Core  
clean-architecture architecture  
Readme  
MIT License

Releases  
4 tags

Sponsor this project  
ardalis Steve Smith  
Sponsor  
Learn more about GitHub Sponsors

# CleanArchitecture Template

---

## Using the dotnet CLI template

---

First, install the template from NuGet:

```
dotnet new -i Ardalis.CleanArchitecture.Template
```

You should see the template in the list of templates from `dotnet new` after this install successfully. Look for "Steve Smith Clean Architecture" with Short Name of "clean-arch".


Navigate to the directory where you will put the new solution.





Run this command to create the solution structure in a subfolder name `Your.ProjectName` :

```
dotnet new clean-arch -o Your.ProjectName
```




The `Your.ProjectName` directory and solution file will be created, and inside that will be all of your new solution contents, properly namespaced and ready to run/test!

# Ardalis.ApiEndpoints




 [ardalis / ApiEndpoints](#)










 Sponsor  Watch 35  Star 656  Fork 48

[Code](#) [Issues 8](#) [Pull requests](#) [Actions](#) [Projects](#) [Security](#) [Insights](#)



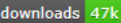
 master  2 branches  8 tags

[Go to file](#) [Code](#)

 JoeyMckenzie and Joey Mckenzie chore(README): Add blog post (#37)  efc0f42 on Sep 29  74 commits

 .github	Update publish.yml	4 months ago
 sample	Example with injection in method instead of constructor (#35)	3 months ago
 src	Update Ardalis.ApiEndpoints.CodeAnalyzers.csproj	4 months ago
 tests/Ardalis.ApiEndpoints.CodeAnaly...	Improve Code Analyzer (#30). Ignore static methods and constructors. (#...	4 months ago
 .editorconfig	Cleaning up and reving package version	4 months ago
 .gitignore	Adding http test commands	10 months ago
 Ardalis.ApiEndpoints.sln	Cleaning up and reving package version	4 months ago
 LICENSE.txt	Moved folders and published to Nuget	10 months ago
 README.md	chore(README): Add blog post (#37)	3 months ago

README.md

 passing  v2.0.0  47k

## About

A project for supporting API Endpoints in ASP.NET Core web applications.

 Readme

 MIT License

## Releases

 8 tags

## Sponsor this project

 ardalis Steve Smith

 Sponsor

[Learn more about GitHub Sponsors](#)

## Packages

# Code Walkthrough

---

[GITHUB.COM/ARDALIS/CLEANARCHITECTURE](https://github.com/ardalis/cleanarchitecture)

# Resources

---

## Clean Architecture Solution Template

<https://github.com/ardalis/cleanarchitecture>

For Worker Services: <https://github.com/ardalis/CleanArchitecture.WorkerService>

Online Courses ([Pluralsight](#) and [DevIQ](#))

- SOLID Principles of OO Design
- N-Tier Architecture in C#
- DDD Fundamentals
- ASP.NET Core Quick Start

<https://ardalis.com/ps-stevesmith>

<https://ardalis.com/ps-stevesmith>

<https://ardalis.com/ps-stevesmith>

<http://aspnetcorequickstart.com/>

## Weekly Dev Tips Podcast

<http://www.weeklydevtips.com/>

Microsoft Architecture eBook/sample  
Group Coaching for Developers

<http://aka.ms/WebAppArchitecture>

<https://devbetter.com/>

# Thanks!

---

Steve Smith

[steve@ardalis.com](mailto:steve@ardalis.com)

@ardalis



**WEEKLY DEV TIPS**

WITH STEVE SMITH (@ardalis)