Clean Architecture

A modern approach to building software

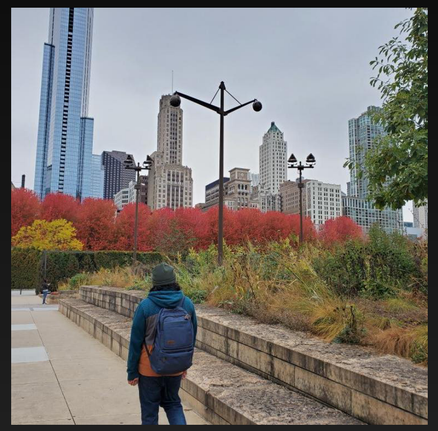
Marco Cabrera - February 22 2024

Joke

- Why is Yoda such a good gardener?
  - Because he has two green thumbs

About the presenter

- Software Developer for over 6 years
- Loves to travel
- Mediocre video game player
- Enjoys Philosophy

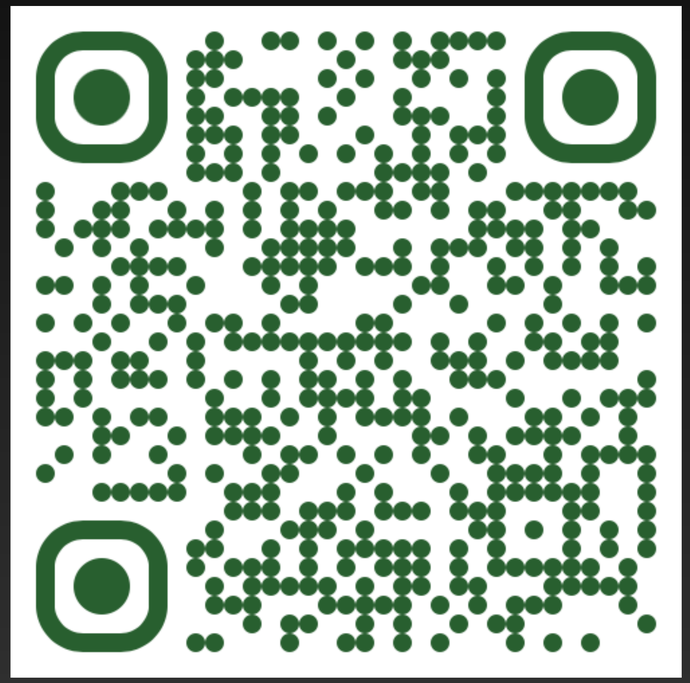


Agenda

- History
- Layers
- Domain Centric Design
- Clean Architecture
- Domain
- Application
- Persistance
- User Interface
- Infrastructure
- Who is this for?
- Pain Points
- Questions

Source Code available on Github

- The code: https://github.com/mck231/CoffeeClub
- Code is Open Sourced



Layers
Layers vs Tiers

- Layers describe the logical groupings of the functionality and components in an application; whereas tiers describe the physical distribution of the functionality and components on separate servers, computers, networks, or remote locations. Although both layers and tiers use the same set of names (presentation, business, services, and data), remember that only tiers imply a physical separation. It is quite common to locate more than one layer on the same physical machine (the same tier). You can think of the term tier as referring to physical distribution patterns such as two-tier, three-tier, and n-tier.

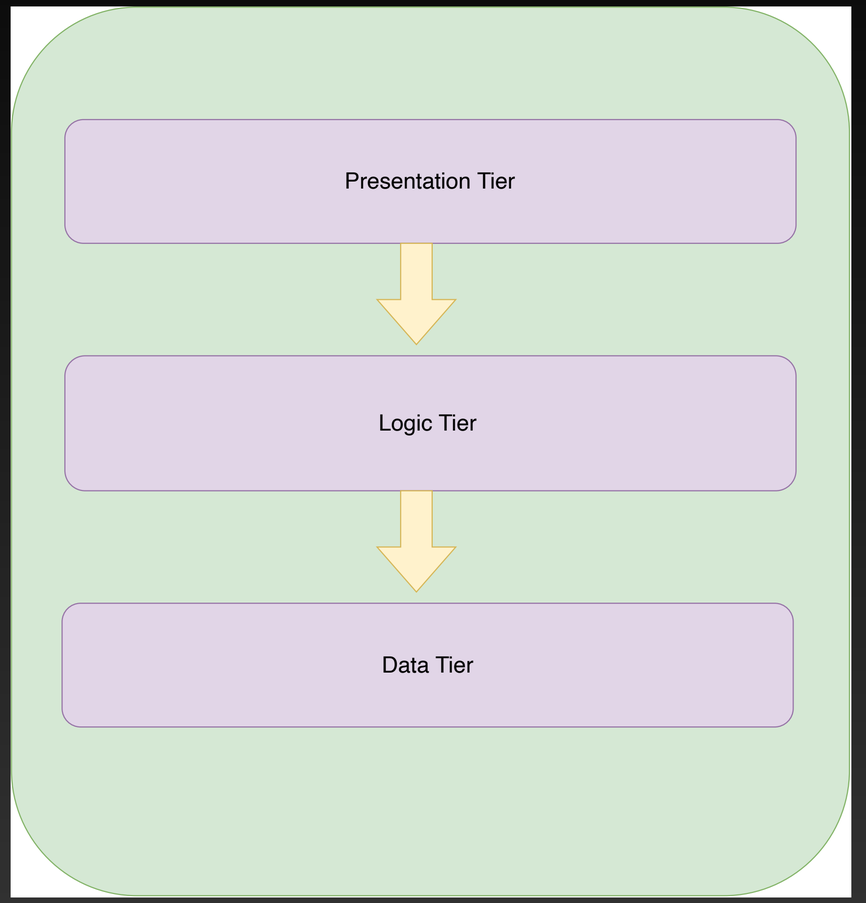
https://learn.microsoft.com/en-us/previous-versions/msp-n-p/ee658109(v=pandp.10)?redirectedfrom=MSDN

History

“Those who cannot remember the past are condemned to repeat it.”

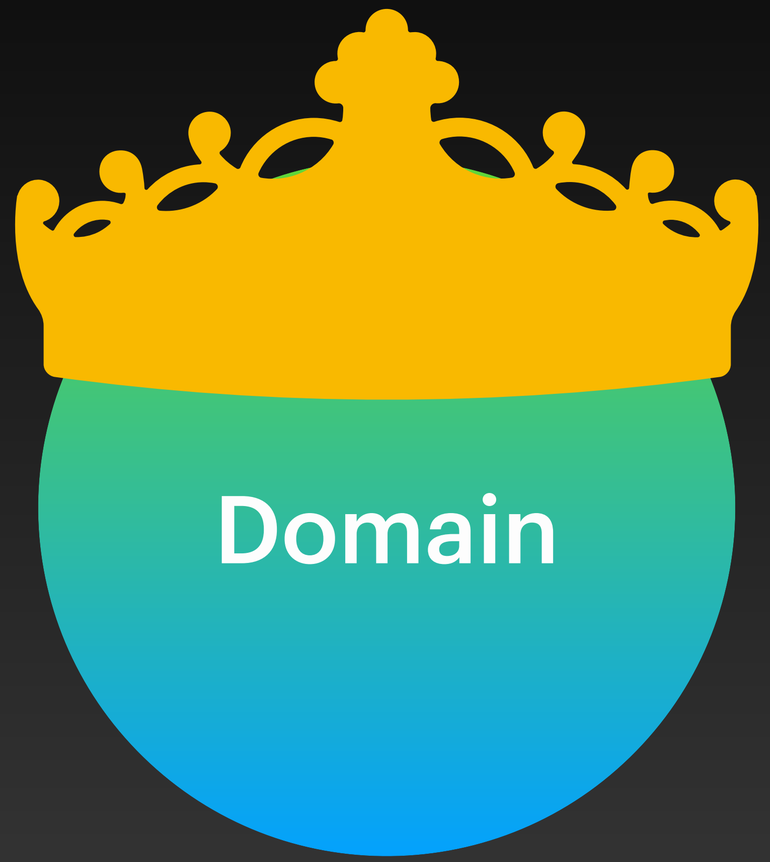
-Jorge Agustín Nicolás Ruiz de Santayana y Borrás

- The N-tier approach
  - Presentation tier - User interface(MVC, Angular, Blazor, React)
  - Middle tier - Business logic, Data Access, Auth
  - Data tier - Database(Microsoft SQL Server)
- Data was the star of the show



Domain Centric Design

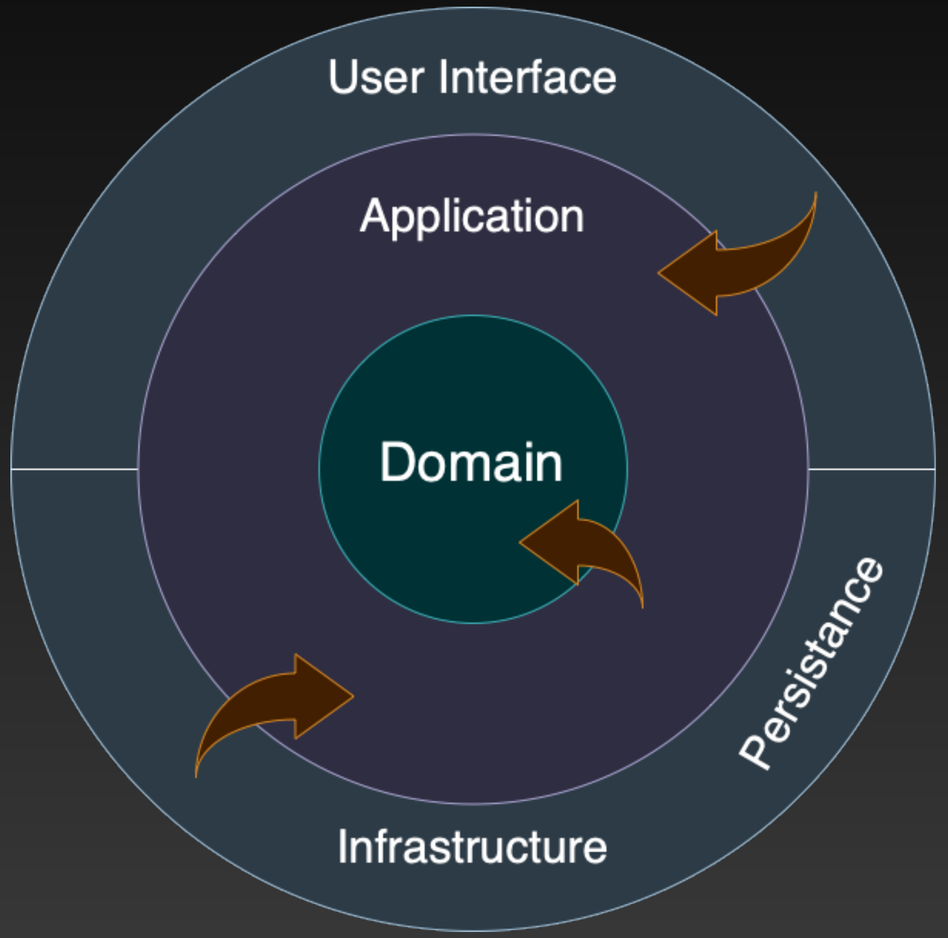
- Domain is now the Star of the show
- Databases now become implementation detail
- Enhances testability by promoting a clear separation of concerns
- Forces Developers and Customers to better know the what is core to the business
- Domain Driven Design



Domain Driven Design book: https://a.co/d/aJDlB4N

Clean Architecture

- Divides software into distinct sections for simplicity and maintainability
- Business logic remains central and unaffected by external changes
- Facilitates technological upgrades, and improves testability



Domain

- The domain layer is where you can express both your business entities and business rules in code. It should utilize a ubiquitous language
- No dependencies on anything; the domain gets referenced but should not reference any other part of the application
- Elements that can be found here include: Entities, Value Objects, Aggregates, Services, and Enums

Application

- This layer holds references to the Domain layer, serving as the bridge between the user interfaces or automated tasks and the core business logic
- The Application layer is responsible for coordinating tasks, whether triggered by a user, a scheduled job (e.g., cron job), or an event. It orchestrates the flow of data and control, delegating to the Domain layer for business processing, or to the Persistence or Infrastructure layers as needed
- Often characterized by its `Use Cases`, this layer includes application-specific features such as `PlaceOrder` or `SubmitAssignment`. These use cases coordinate with other layers to fulfill application operations
- The Application layer may contain components such as Command Handlers, Query Handlers, Service Interfaces, and Application Services that facilitate the execution of application logic

Outer Layers

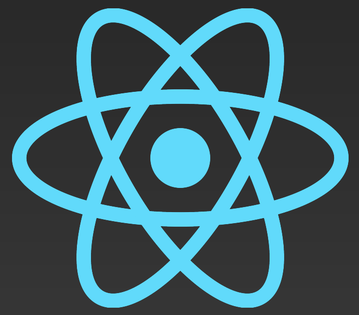
- The User Interface ex: Blazor, Angular, ASP.NET MVC, HTML and Vanilla JS
- The Persistence Layer
- The Infrastructure Layer: CSV export, PDF export, or Email Service

Persistance Layer

- This is houses database-specific code and data storage mechanisms
- Repositories can serve as logic to commit database transactions. They act as mediators between the Domain layer and data mapping layers, encapsulating the logic required to access data sources
- The DBContext can be found here in this layer, which acts as the liaison with the database, It facilitates the querying and saving of entity instances; and can also serve as a Initial seeding point for database data.
- This layer treats databases as an implementation detail, effectively decoupling the persistence mechanisms from the Domain logic

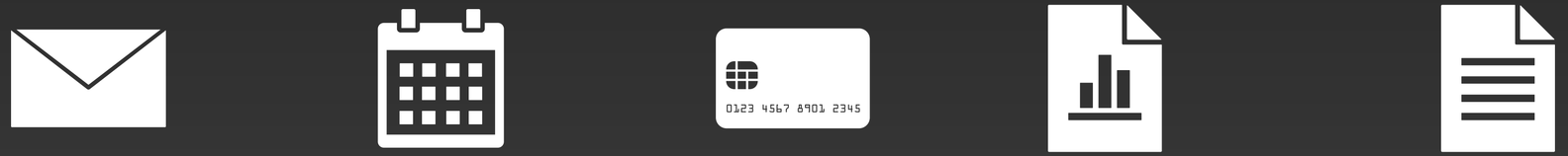
User Interface

- Is how your users interact with the application
- Could be a mobile app, Website, or anything in between
- Examples: Blazor, Angular, Vue, React, ASP.NET Core MVC
- This also becomes a implementation detail in Clean Architecture



Infrastructure Layer

- Contains things not having to do with the application but perhaps needed
- Examples: Email Sender, PDF exporter, CSV exporter, File manipulation, other third party services. Maybe this could be composing a prompt to send to OpenAI



Who is this for?

- For projects where requirement could rapidly change
- For when you need high test coverage
- For teams that want to work more closely with business
- For teams thinking of moving into a micro service architecture
- For larger teams, work can be divided more easily

Pain

- Sometimes it’s not obvious where things go(like DTOs)
- Lots of boilerplate code
- Sometimes you have to Repeat yourself
- Sometimes you need have a dependency in your Domain layer(MongoDB)
- Steep learning curve

Questions?