ASP.NET Core

-

SOLID and Clean Architecture

HELLO!

I am Trevoir Williams

Software Engineer | Lecturer

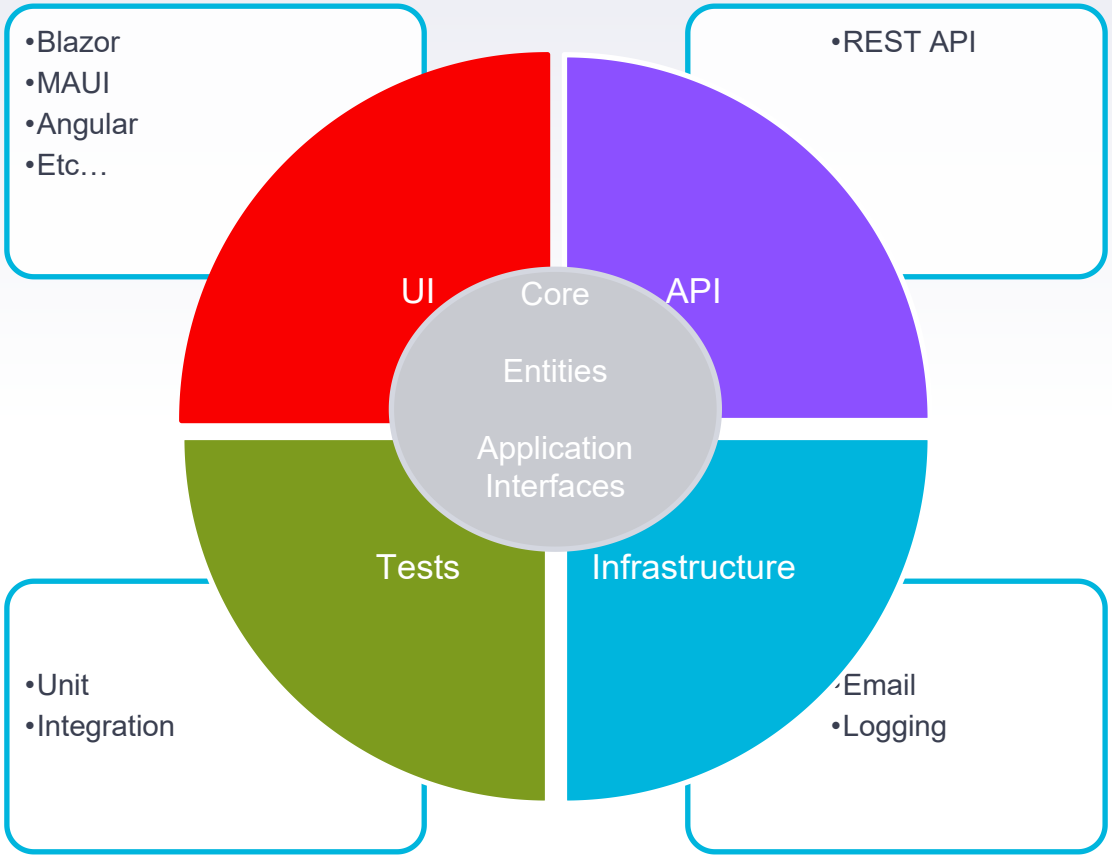
Course Objectives

Understand Clean Architecture

- Learn SOLID Principles
- Clean Architecture in ASP.NET Core
- Advanced Concepts
  - CQRS and Mediator patterns
  - Fluent Validation
  - Global Exceptions and Logging
- Unit and Integration Testing
- API Development
- Blazor Development

Learn Development Tools

- Learn How To Use:
  - Visual Studio 2022
  - .NET 7 SDK
  - Entity Framework Core
  - ASP.NET Core Web API
  - NSwagStudio
  - Blazor WebAssembly


•Blazor
•MAUI
•Angular
•Etc…
•REST API
UI
API
Core
Entities
Application
Interfaces
Tests
Infrastructure
•Unit
•Integration
•Email
•Logging

Prerequisites

- Visual Studio 2022 and .NET 7
- C#/.NET Programming Knowledge
  - Object Oriented Programming
  - Entity Framework

Learning Outcomes

- Understand Clean Architecture in ASP.NET Core
- Understand and Apply SOLID Principles .NET Core Development
- Know how to implement CQRS and Mediator patterns
- Build API and consume in Blazor

Welcome!

Let’s get Started

Development Environment

- Visual Studio 2022
- .NET 7 SDK (.NET 6 SDK compatible)
  - REST API
  - Blazor
  - Entity Framework Core

S.O.L.I.D Principles

- S - Single-responsibility Principle
- O - Open-closed Principle
- L - Liskov Substitution Principle
- I - Interface Segregation Principle
- D - Dependency Inversion Principle

Separation Of Concerns & Single Responsibility

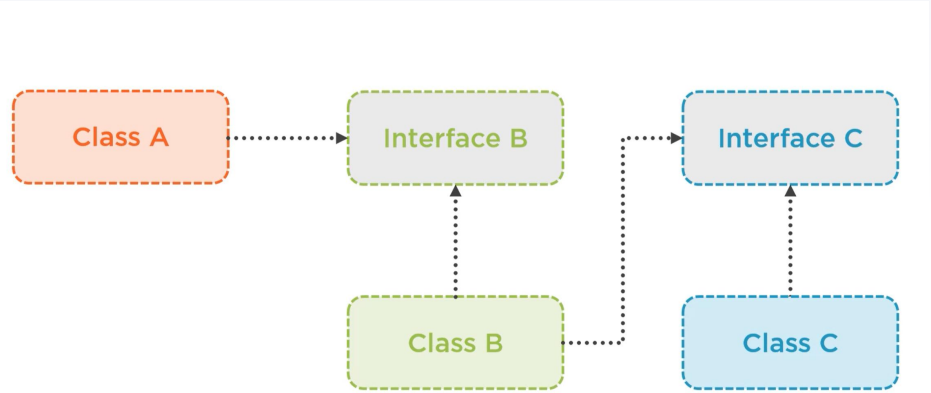
- S in SOLID
- Foundation of Object Oriented Programming
- Concept of splitting functionality into blocks
  - Each addressing a specific concern
  - One block of code shouldn't be trying to do many different things
- Promotes Modularity
  - Each Module encapsulates all logic for the specific feature set.

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.
- 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
  - 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

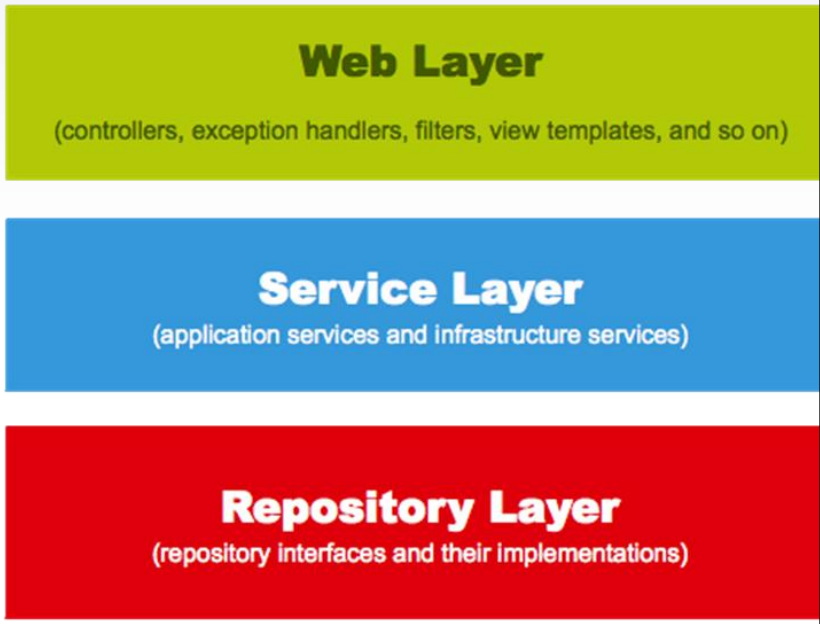
Layered Architecture

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



Credit: Martin Ledvinka

Understanding Clean Architecture

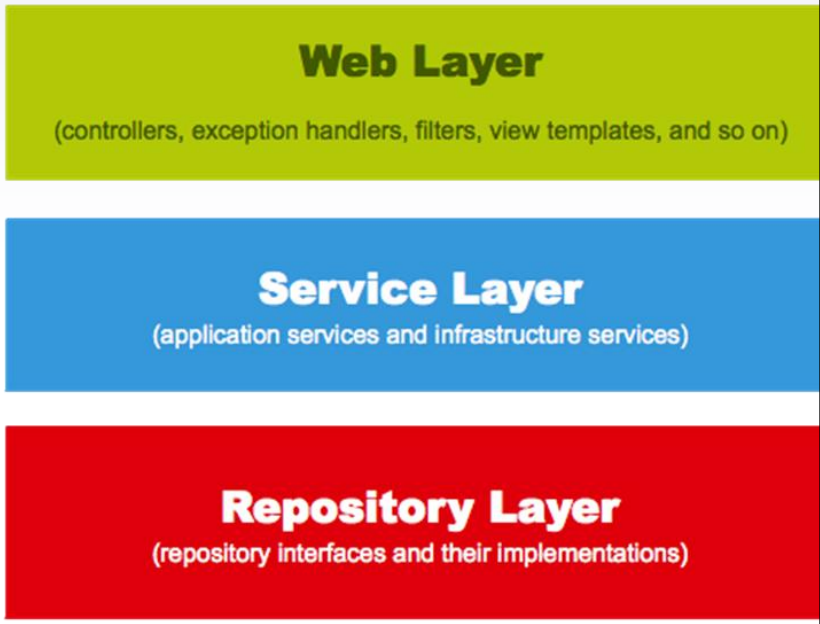
Layered Architecture

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

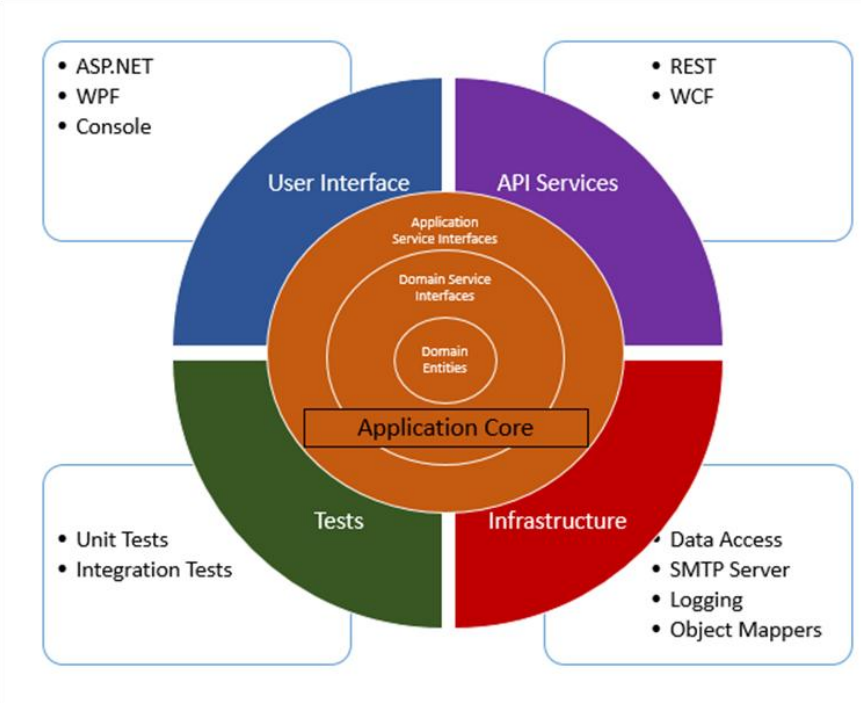


Credit: Martin Ledvinka

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



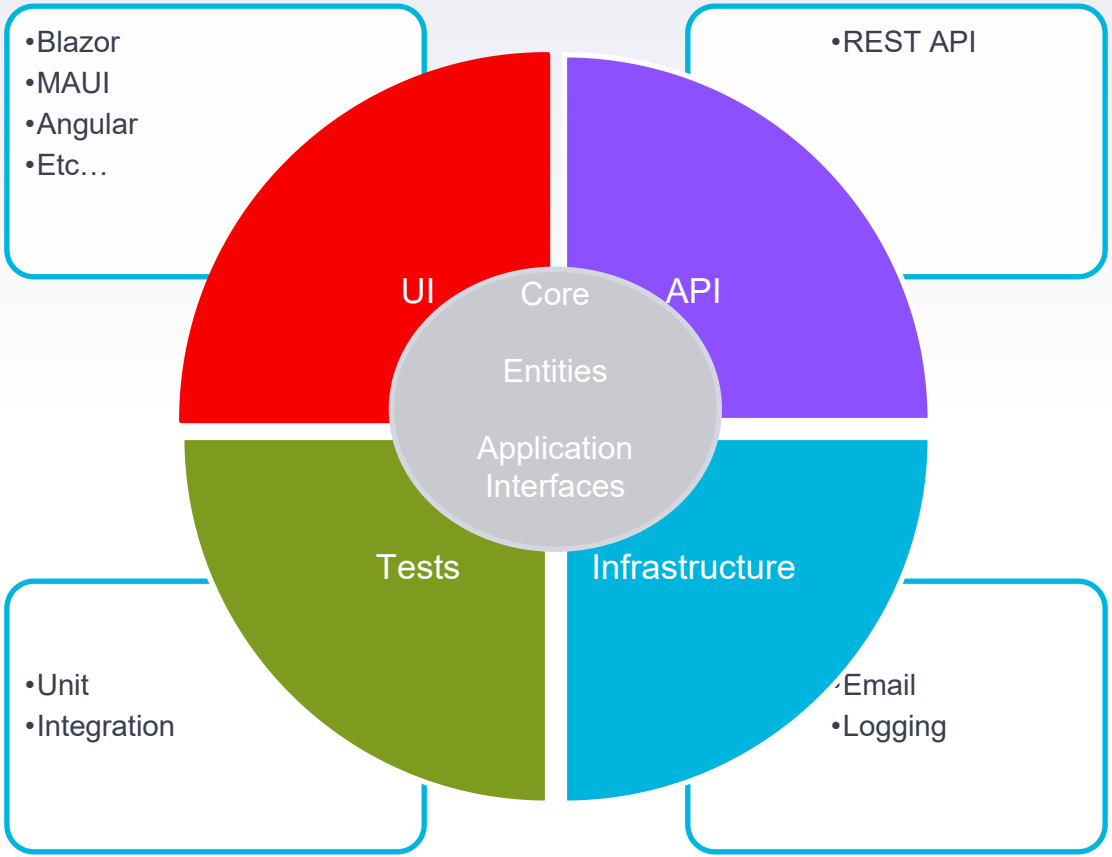
Credit: Lori Peterson

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.

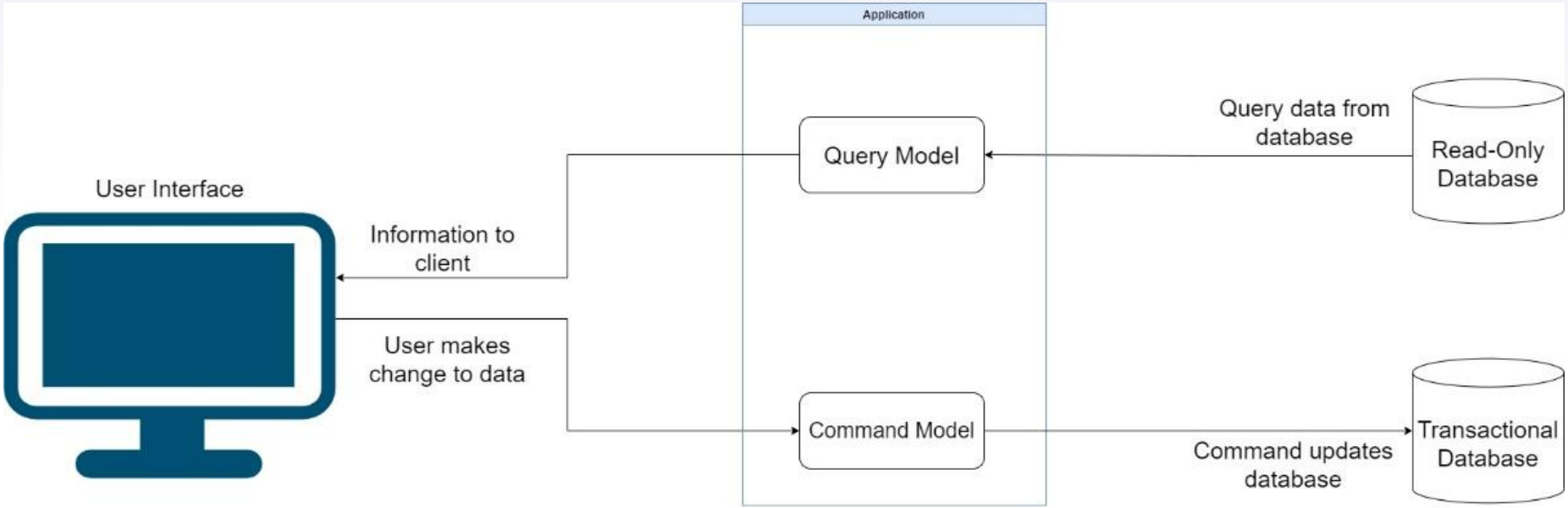
- What is the scale of the application?
  - Not every project needs 'Clean Architecture' from day one.
  - Start small and extend as needed.


•Blazor
•MAUI
•Angular
•Etc…
•REST API
UI
API
Core
Entities
Application
Interfaces
Tests
Infrastructure
•Unit
•Integration
•Email
•Logging

Automapper

- Converts complex data types with ease
- Can be used in several parts of the application
- Saves time spent on writing manual mapping code
- Might lead to performance and debugging issues

CQRS Pattern



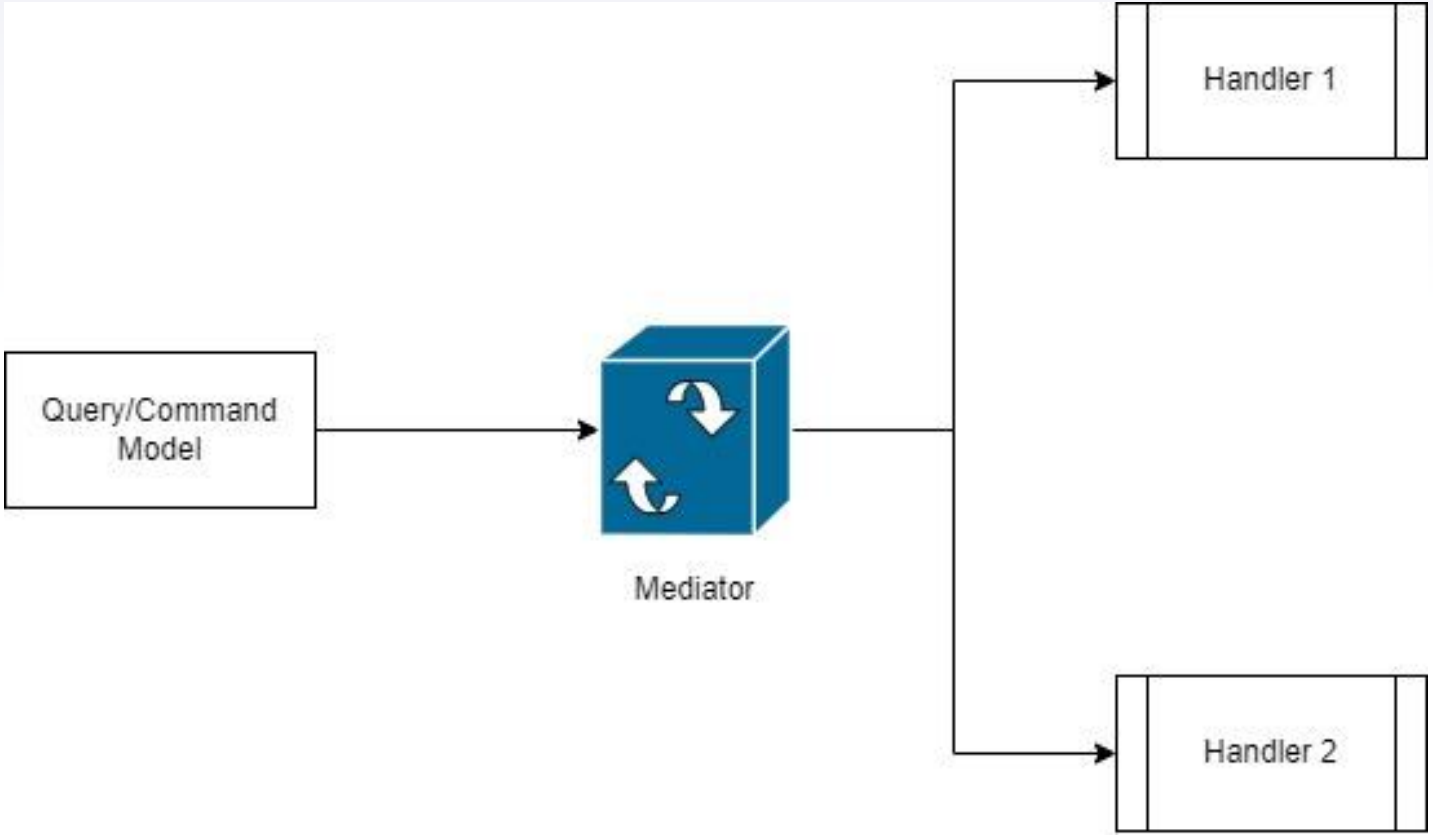
Example Request

```
public record GetLeaveTypesQueryRequest ():
IRequest<List<LeaveTypeDto>>;
```

Example Handler

```
public class GetLeaveTypesRequestHandler : IRequestHandler<
GetLeaveTypesRequest, List<LeaveTypeDto>>
{
  public GetLeaveTypesRequestHandler(/* Dependencies */)
   public async Task<List<LeaveTypeDto>
Handle(GetLeaveTypesRequest request)
        {
            return await _repo.GetAsync();
        }
    }
```

Mediator Pattern



Mediatr

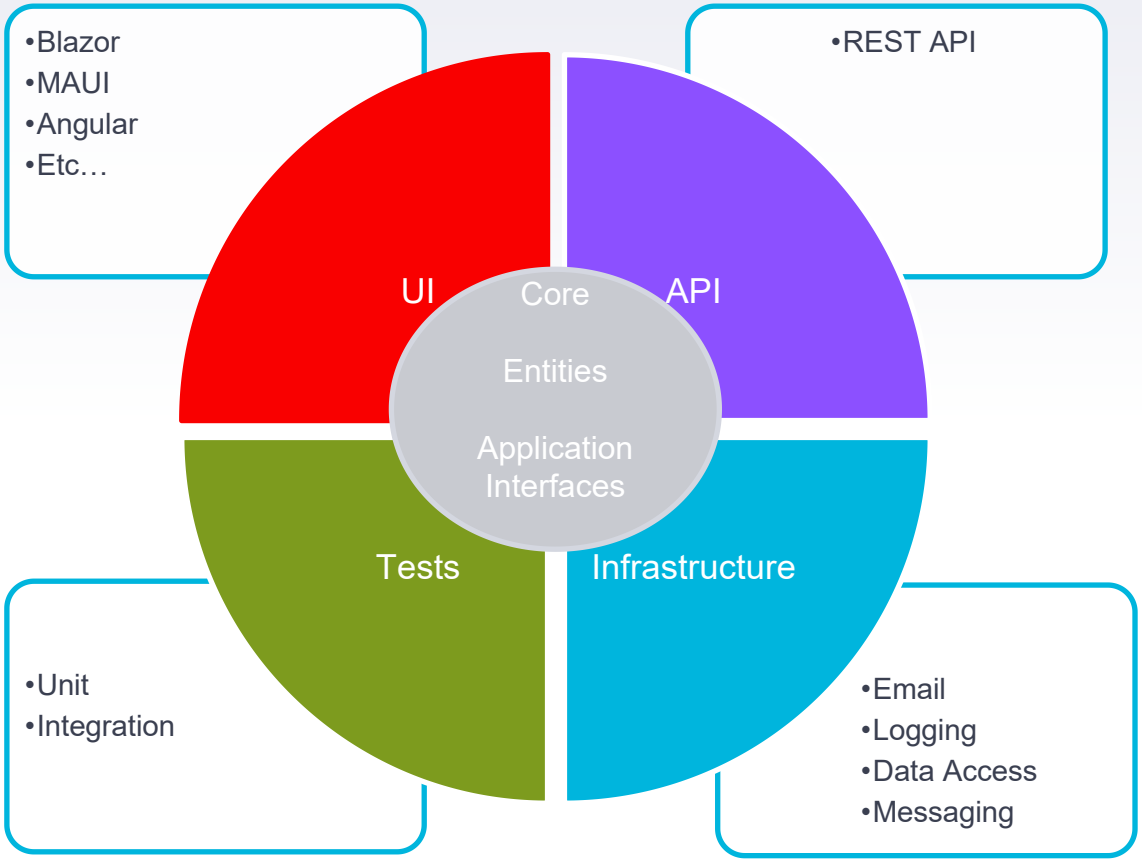
```
private readonly IMediator _mediator;
public APIController(IMediator mediator) => _mediator
= mediator;

public async Task<ActionResult<List<LeaveTypeDto>>>
Get()

{

   var leaveTypes = await _mediator.Send(new
GetLeaveTypesQueryRequest());

            return Ok(leaveTypes);

}
```

Section Review

- Setup Application Core
  - Domain and Application Projects
- Added MediatR and Automapper
  - CQRS – Requests and Handlers
  - Object Mapping
- Fluent Validation
  - Complex but flexible validation logic

Infrastructure Layer

- Implementations of contracts
  - E.g. Repository Implementations
- Entity Framework Core
- Third-Party Services
  - Email
  - Logging
  - Messaging


•Blazor
•MAUI
•Angular
•Etc…
•REST API
UI
API
Core
Entities
Application
Interfaces
Tests
Infrastructure
•Unit
•Integration
•Email
•Logging
•Data Access
•Messaging

Logging

- Logs are blocks of text that most applications produce during runtime.
- Human readable mini reports about what is happening in the application
- Allow us to be able to track and trace errors that occur in our applications.
  - **Information** – standard log level used when something has happened as expected.
  - **Debug** – a very informational log level that is more than we might need for everyday use.
  - **Warning** – this log level indicates that something has happened that isn't an error but isn't normal.
  - **Error** – an error is an error. This type of log entry is usually created when an exception is encountered.

Section Review

- Setup Persistence Project
  - Added EF Core
  - Implemented Repositories
- Added Infrastructure Project
  - For Third-Party services
  - Options Pattern
  - Email, Logging, etc...

Section Overview

- Develop API
  - Review Dependency Injection
  - Create Database using EF Core commands
  - Develop Thin Controllers using MediatR
  - Implement remaining features

Section Review

- Configured API
  - Injecting project dependencies
  - Create Database using EF Core commands
  - Implemented Thin Controllers
  - Setup Global Exception Handling
  - Add application configurations (email and database)
- Added Leave Allocation and Leave Request Features

Section Overview

- Automated Tests
  - Unit and Integration Testing
  - Xunit – Testing Framework
  - Moq – Mocking framework
  - Shouldy – Assertions based on Fluent API

Section Review

- Automated Tests
  - Validate that code operates as expected
  - Tests known or expected outcomes
  - Best implemented when code is clean and modular
  - Easier to accomplish when code is loosely coupled
- Integration and Unit Tests
  - Unit Tests validate scenarios for methods and operations
  - Integrations tests validate behaviours of third-party frameworks and technologies
- Challenge: Write more Tests!

Section Overview

- Adding the UI
  - Blazor - .NET Single Page Application development framework
  - NSwag and NSwagStudio – Generate API integration code
  - Configure JWT security
    - Admin and Employee Roles
  - Setup UI for Leave management features

Section Review

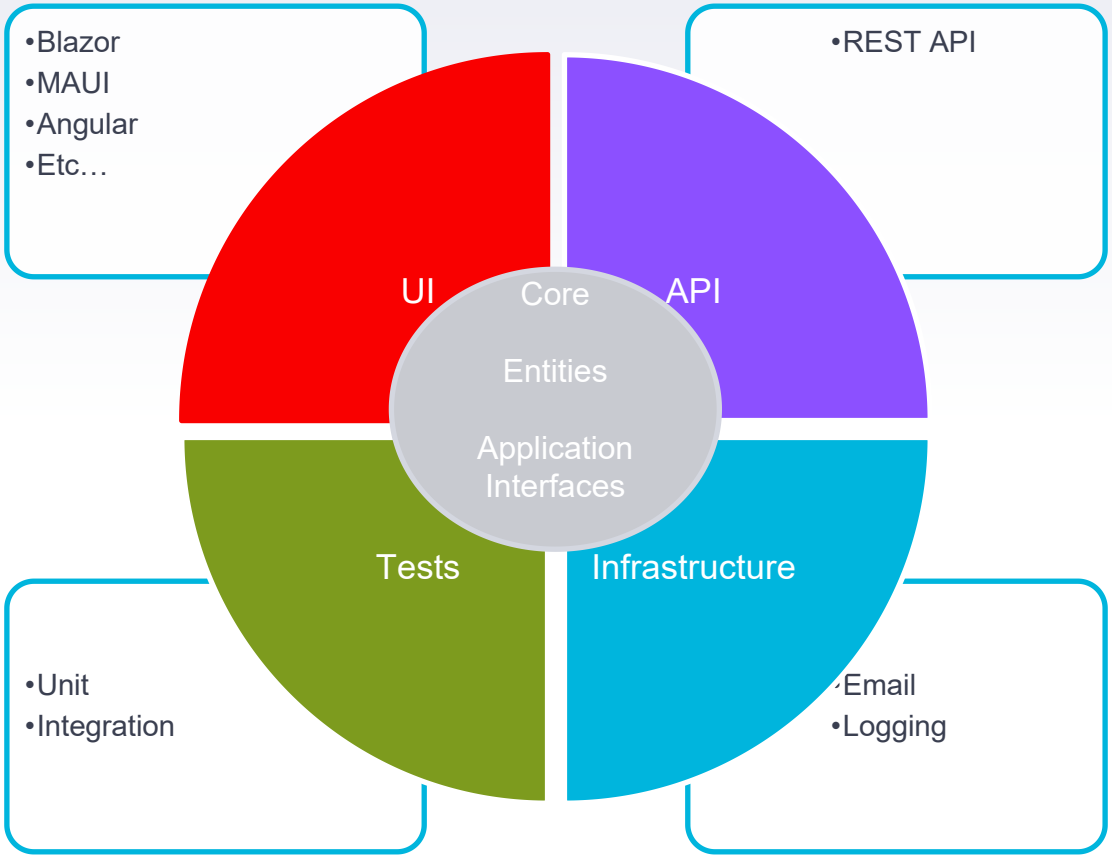
- Automated Tests
  - Validate that code operates as expected
  - Tests known or expected outcomes
  - Best implemented when code is clean and modular
  - Easier to accomplish when code is loosely coupled
- Integration and Unit Tests
  - Unit Tests validate scenarios for methods and operations
  - Integrations tests validate behaviours of third-party frameworks and technologies
- Challenge: Write more Tests!

Conclusion

- Clean Architecture
  - Application & Infrastructure Layers
  - Unit and Integration Testing
  - API Development (Response Logging, JWT Security, Swagger Documentation)
  - Blazor UI Development (Consume API, NSwagStudio, Custom Handlers, and Auth Provider)
- Coding Patterns and Concepts
  - Dependency Injection
  - Separation of Concerns
  - CQRS and Mediator Patterns
  - Repository Pattern

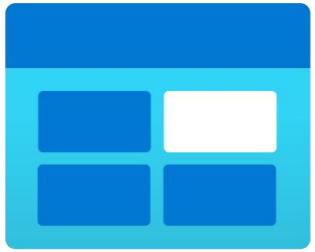
THANK YOU

Trevoir Williams


•Blazor
•MAUI
•Angular
•Etc…
•REST API
UI
API
Core
Entities
Application
Interfaces
Tests
Infrastructure
•Unit
•Integration
•Email
•Logging

Other Technologies

We will interact with quite a few Azure Services along the journey.



Azure Storage

Azure AD

Azure CosmosDB

Azure Application Insights

Azure App Services

Section Review

Blazor WebAssembly

- Used to create a simple client app to interface with our API
- Used NSwagStudio to generate API client code
- Custom Components and clean coding practices

API Security

- Setup Identity Services and Identity Tables
- Secured API with JWT
- Secure communication between Blazor and API

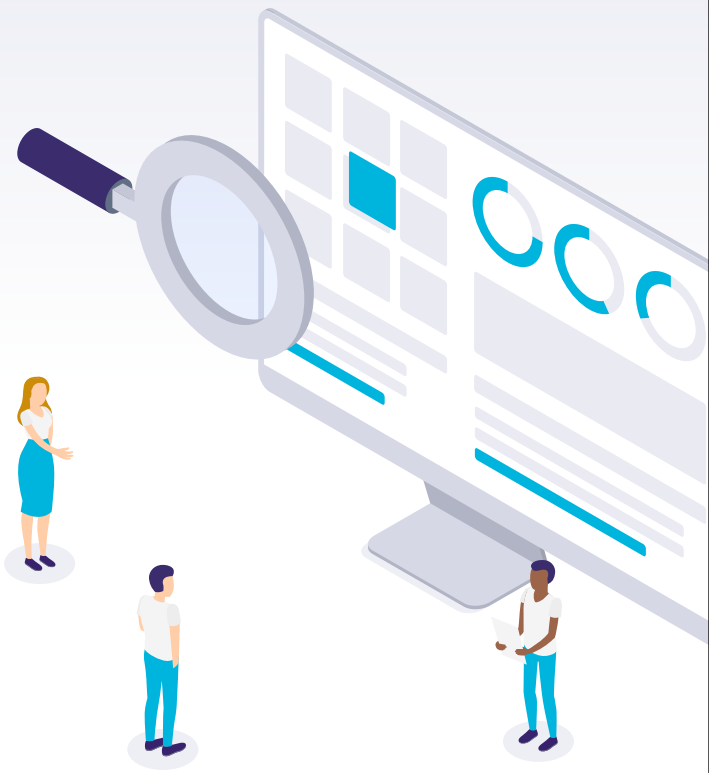
Visual Studio 2022 and .NET 6

What

A fully-featured, extensible, free IDE for creating modern applications for most devices and platforms.

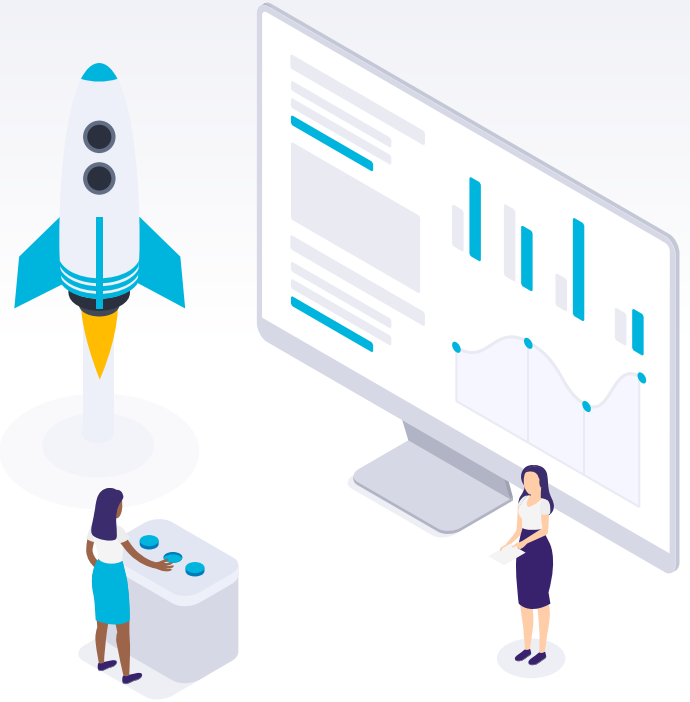
Why

- Powerful Web Tools to allow development with ASP.NET, jQuery, Bootstrap, among other popular frameworks.
- Git Integration - Manage your source code in Git repos hosted by any provider, including GitHub.



1

Why Learn .NET Core



- Microsoft .NET is the platform that drives the business technology of many of the top corporations in the United States and many other countries.
- It is the predominant technology used to drive enterprise-scale business technology.

Why Learn .NET Development

- The .NET language of choice to learn is C#, the most widely used language today.
- It is a general purpose programming language that can handle almost any problem, from desktop to mobile to dynamic web applications.
- There is a high demand across the world for .NET developers in a variety of industries.

Publish Your App

- Users need access
- Internal or External
- Options
  - Domain Hosting
  - Internal Server
  - Cloud Hosting (Azure, etc.)
- Continuous Integration and Deployment
  - GitHub Actions
  - Azure DevOps

Get Ready To Rumble!