MVC Concepts

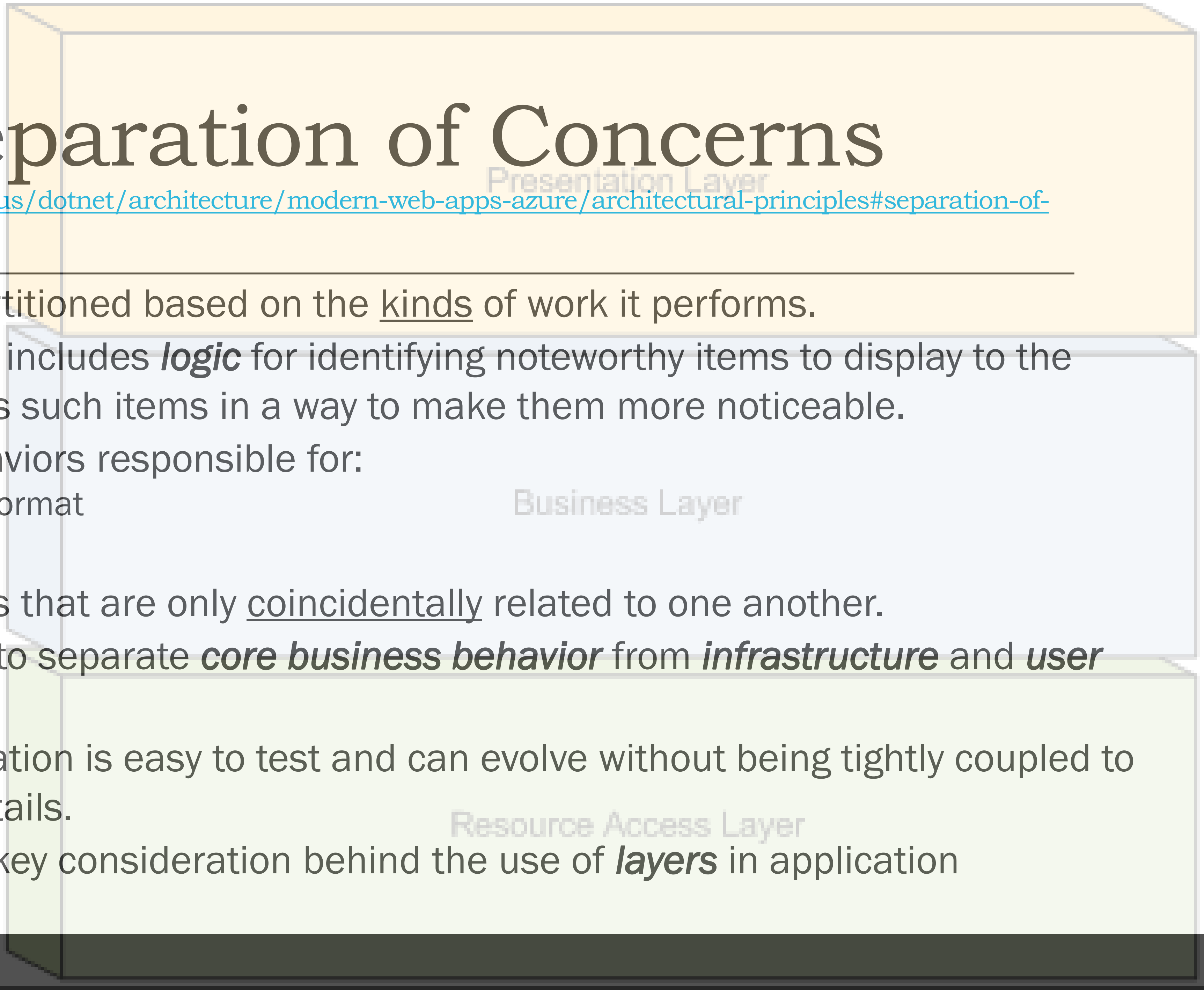
.NET

*The **Model-View-Controller (MVC)** architectural pattern separates an application into three main groups of components: **Models**, **Views**, and **Controllers**. This pattern helps to achieve separation of concerns.*

HTTPS://DOCS.MICROSOFT.COM/EN-US/ASPNET/CORE/MVC/OVERVIEW?VIEW=ASPNETCORE-3.1

MVC - Separation of Concerns

https://docs.microsoft.com/en-us/dotnet/architecture/modern-web-apps-azure/architectural-principles#separation-of-concerns



An application should be partitioned based on the <u>kinds</u> of work it performs.

Consider an application that includes *logic* for identifying noteworthy items to display to the user, and which then formats such items in a way to make them more noticeable.

There are two separate behaviors responsible for:

1) choosing which items to format
2) formatting the items.

These are separate concerns that are only <u>coincidentally</u> related to one another.

Applications should be built to separate ***core business behavior*** from ***infrastructure*** and ***user interface*** logic.

This ensures that the application is easy to test and can evolve without being tightly coupled to low-level implementation details.

Separation of concerns is a key consideration behind the use of ***layers*** in application architectures.

MVC – Control/Data Flow

https://docs.microsoft.com/en-us/aspnet/core/mvc/overview?view=aspnetcore-3.1#what-is-the-mvc-pattern

The *Model-View-Controller (MVC)* architectural pattern separates an application into *Models*, *Views*, and *Controllers*.

1. User requests are routed to a *Controller* which
2. works with the *Model* to perform user actions and/or retrieve results of queries.
3. The *Controller* then chooses the appropriate *View* and
4. provides it with the *Model* data it requires to display results to the user.

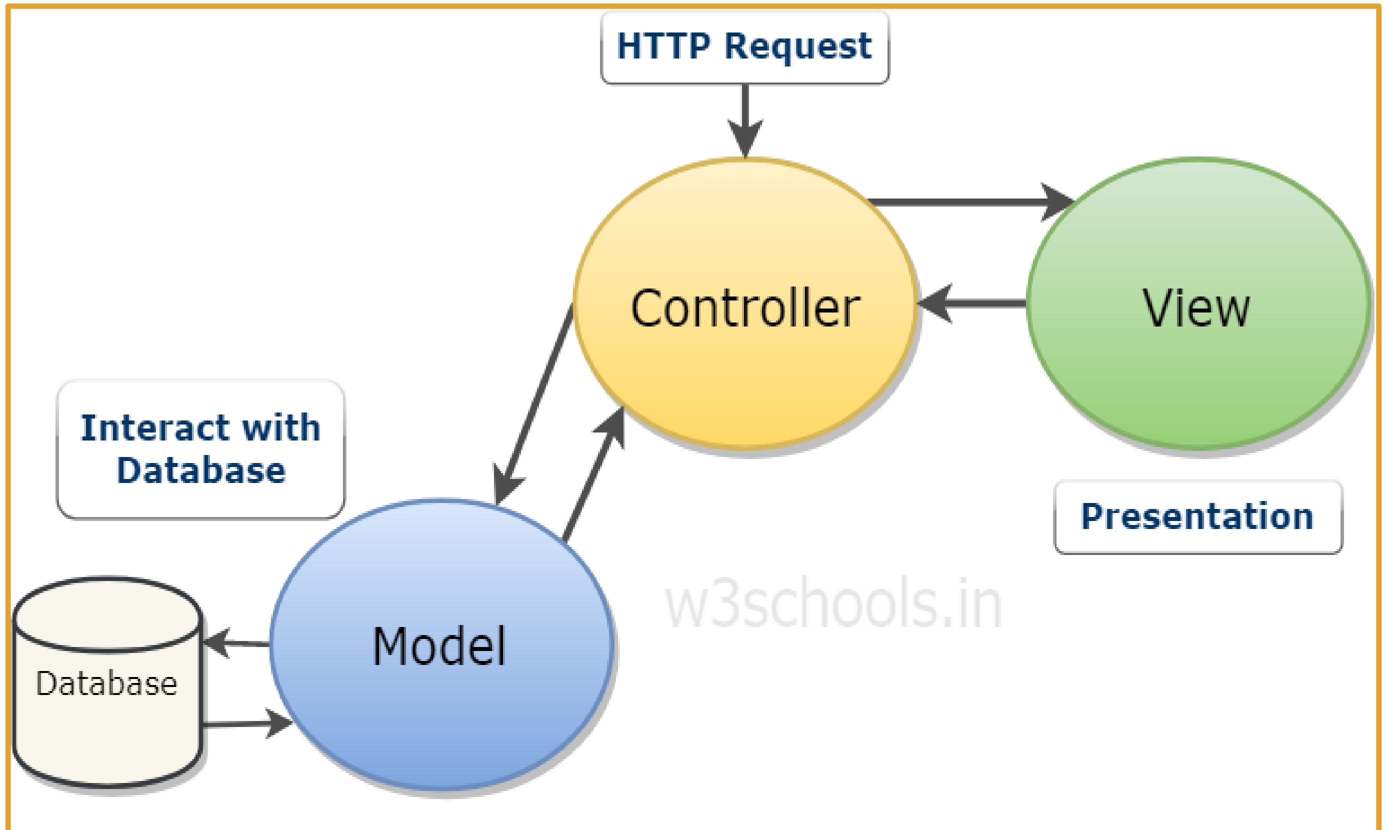


Fig: MVC Architecture

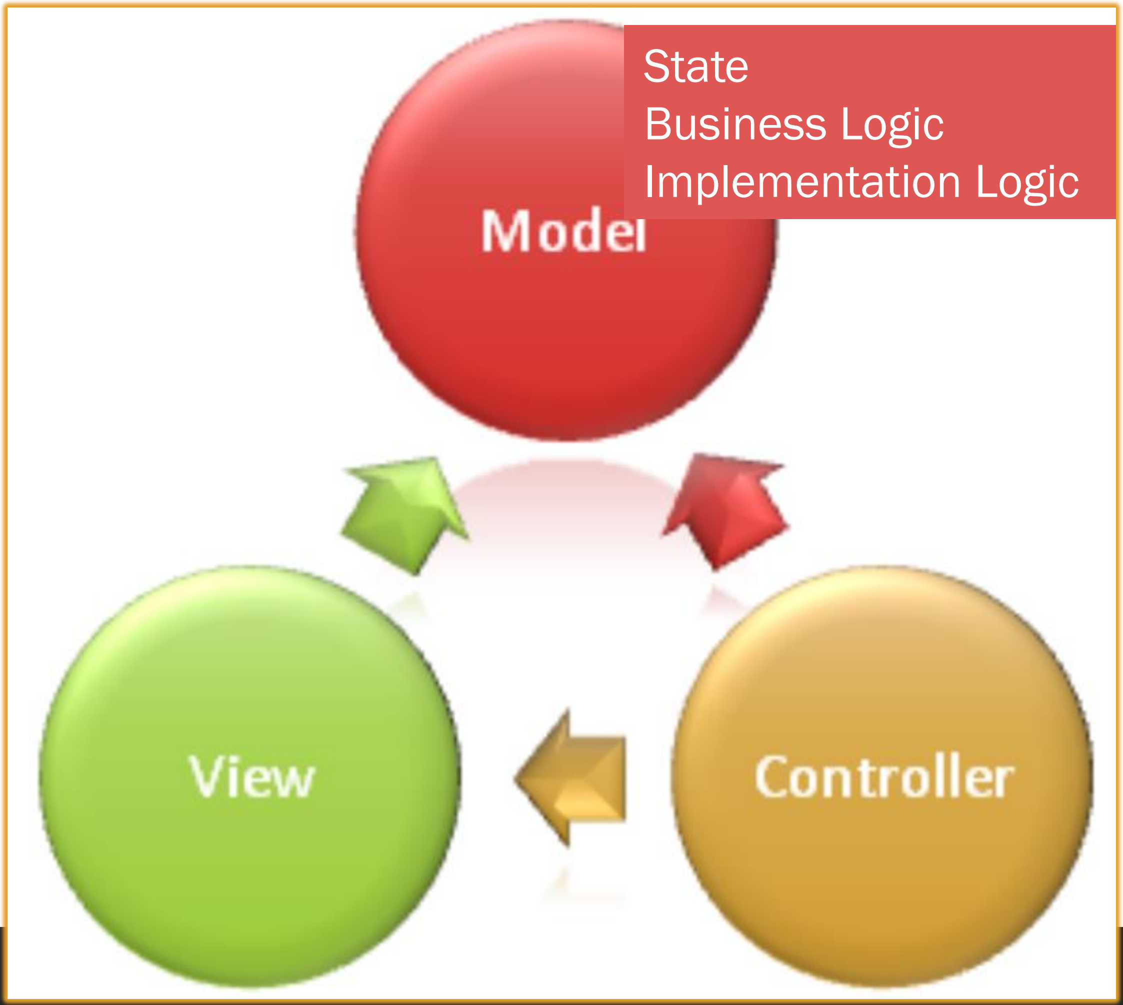
MVC – Model

https://docs.microsoft.com/en-us/aspnet/core/mvc/overview?view=aspnetcore-3.1#model-responsibilities

The ***Model*** part of an ***MVC*** application represents the state of the application and any business logic or operations that should be performed by it.

Business logic is encapsulated in the ***Model***, along with any implementation logic (DbContext) for persisting the state of the application (the DataBase).

Strongly-typed ***views*** typically use ***ViewModel*** types designed to contain the data to display on that ***view***. The controller creates and populates these ***ViewModel*** instances from the ***model***.



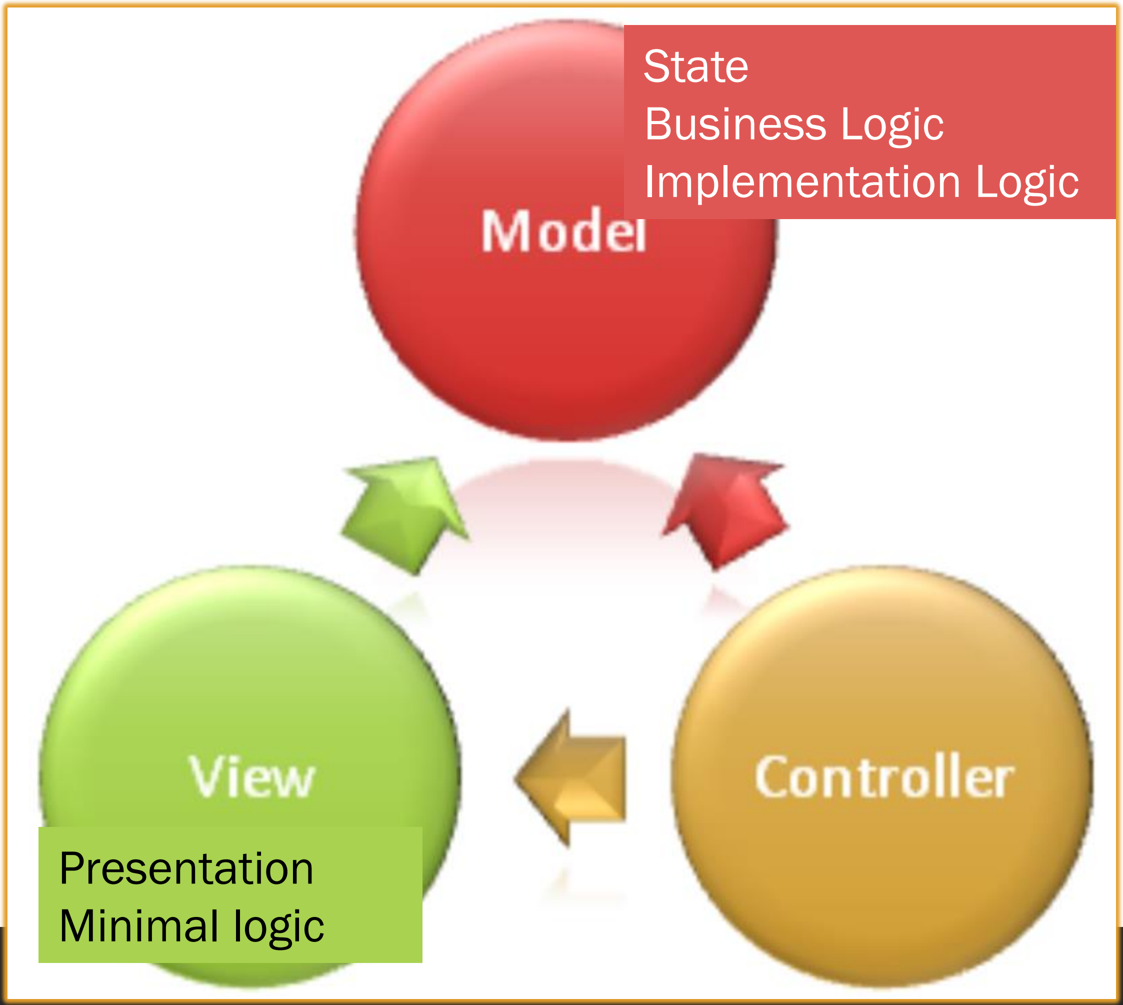
MVC – View

https://docs.microsoft.com/en-us/aspnet/core/mvc/overview?view=aspnetcore-3.1#model-responsibilities

Views are responsible for presenting content through the user interface.

Views use the ***Razor view engine*** to embed .NET code in HTML markup.

Logic in ***Views*** should relate to presenting content. If logic is necessary in order to display data from a complex model, use a ***View Component*** or ***ViewModel*** simplify the view.



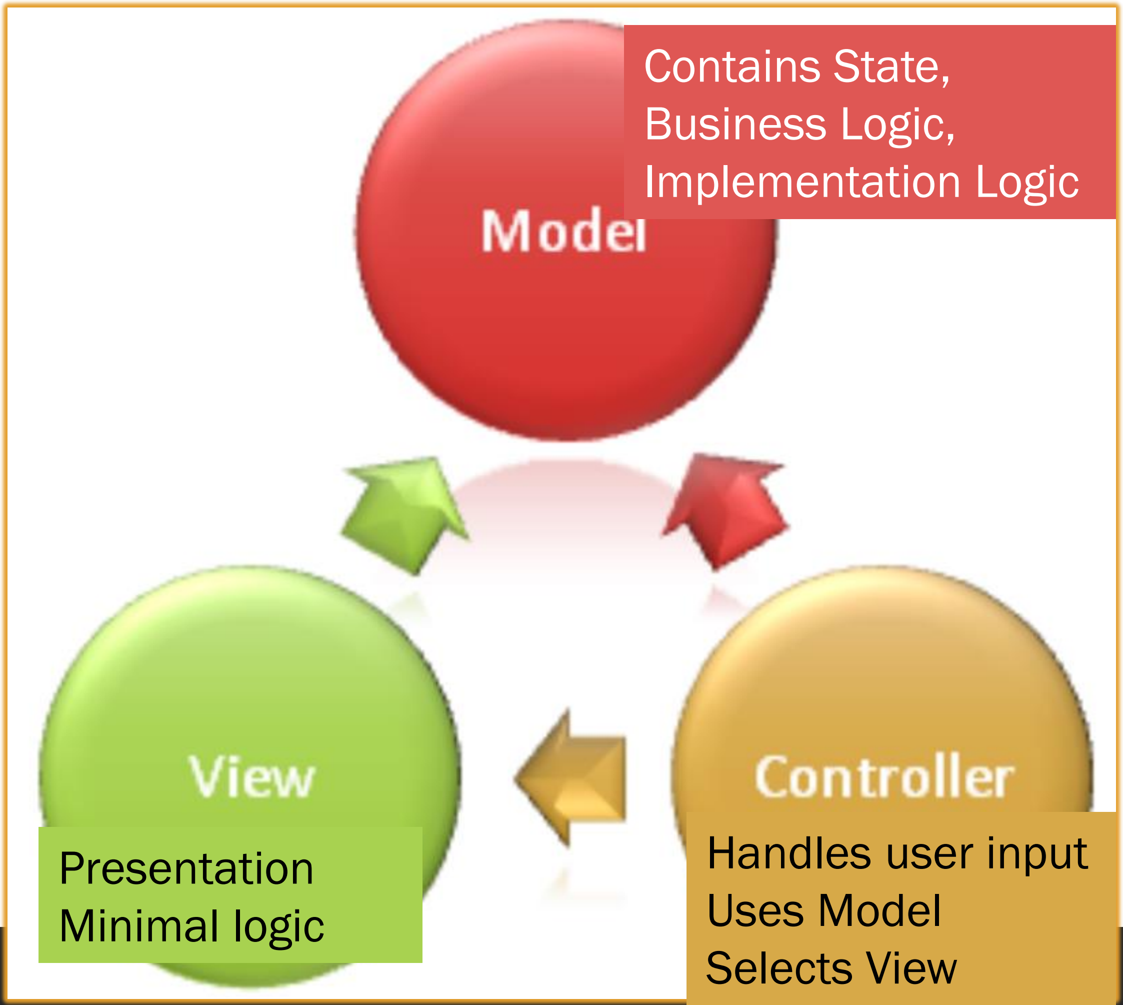
MVC – Controller

https://docs.microsoft.com/en-us/aspnet/core/mvc/overview?view=aspnetcore-3.1#model-responsibilities

Controllers are the components that

- handle user input,
- work with the ***model***, and
- select a ***view*** to render.

The controller handles and responds to user input. In the MVC pattern, the controller is the initial entry point, and is responsible for selecting which model types to work with and which view to render (hence its name - it controls how the app responds to a given request).



Why Use ASP.NET Core MVC?

Use of the MVC approach helps you create applications that separate the different aspects of your application (input logic, business logic, and UI logic), while providing a loose coupling between these elements. The pattern specifies where each kind of logic should be located in the application.

This architecture is good but what about testing? What about dynamic web pages? What about model validation and Web API's? Wouldn't it be nice to have a framework with all those technologies built in?

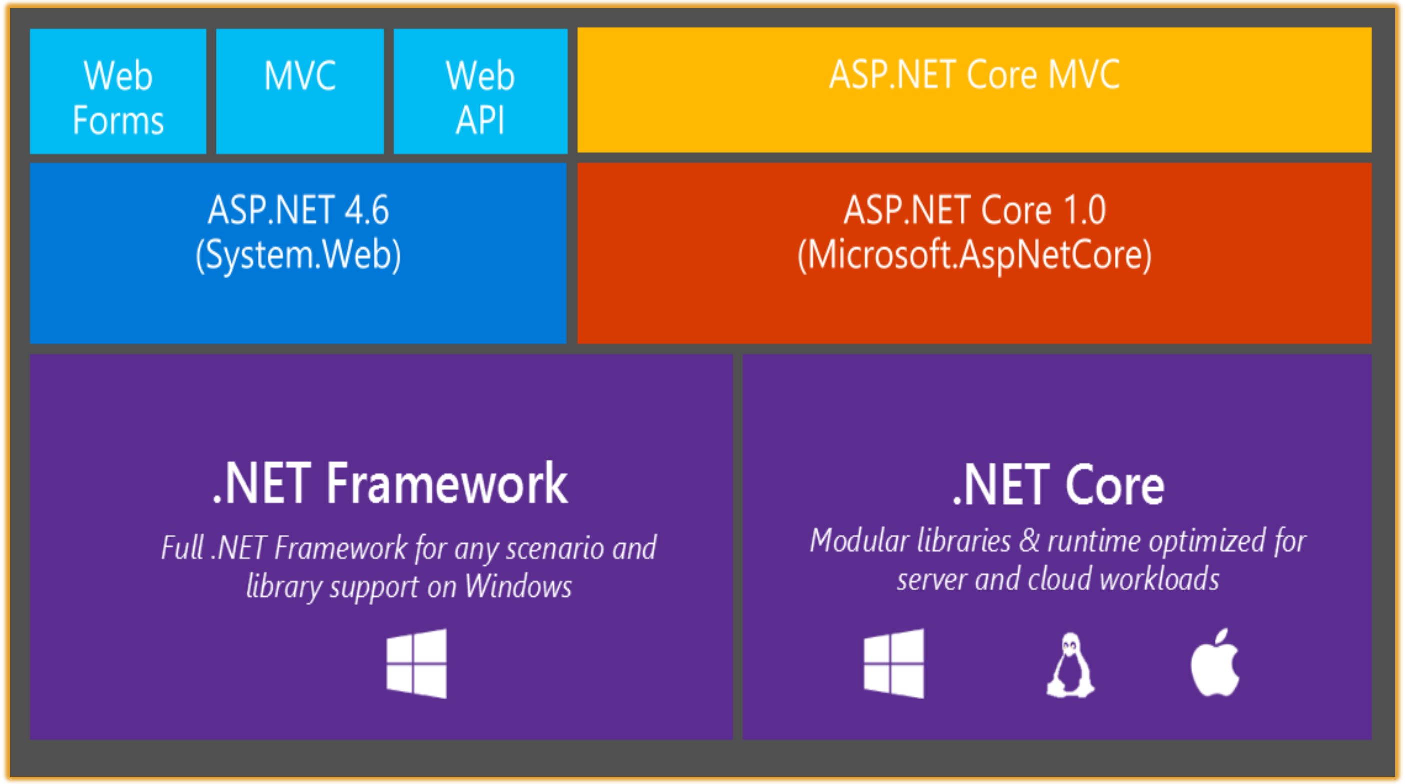
This is where ASP.NET Core MVC comes into the picture.

ASP.NET Core MVC - Overview

https://docs.microsoft.com/en-us/aspnet/core/mvc/overview?view=aspnetcore-3.1&source=docs#what-is-aspnet-core-mvc

The ***ASP.NET Core MVC*** framework is a lightweight, open source, highly testable presentation framework optimized for use with ***ASP.NET Core*** but using the MVC architectural pattern..
ASP.NET Core MVC provides a patterns-based way to build dynamic websites that enables a clean separation of concerns.
ASP.NET Core MVC:

- full control over markup,
- supports TDD-friendly development
- uses the latest web standards.


Web Forms
MVC
Web API
ASP.NET Core MVC
ASP.NET 4.6 (System.Web)
ASP.NET Core 1.0 (Microsoft.AspNetCore)
.NET Framework
Full .NET Framework for any scenario and library support on Windows
.NET Core
Modular libraries & runtime optimized for server and cloud workloads