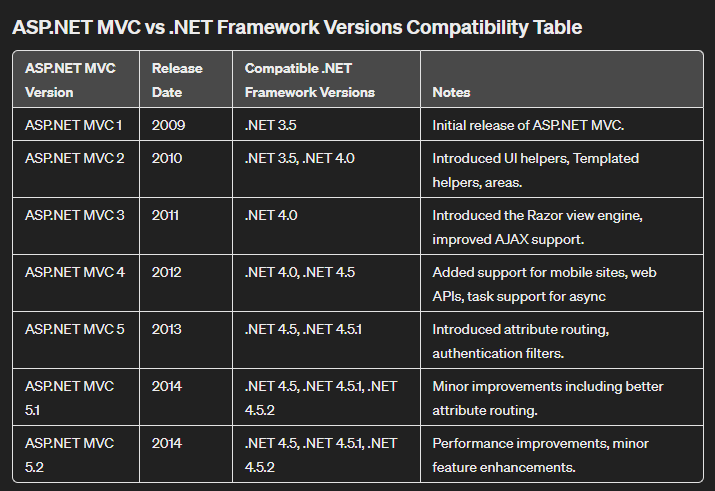
# MVC

### Version history



### Views?

### Partial Views?

### What is view engine

### IActionResult?

### Ways to pass data to views?

Strongly typed data

**viewmodel**

Weakly typed data

**ViewData (ViewDataAttribute)**

**ViewBag**

### Strongly vs weakly types models

Strong typing (or strongly typed) means that every variable and constant has an explicitly defined type (for example, string, int, or DateTime). The validity of types used in a view is checked at compile time.

Weak types (or loose types) means that you don't explicitly declare the type of data you're using. You can use the collection of weakly typed data for passing small amounts of data in and out of controllers and views.

#### Strongly typed - viewmodel

In view, Specify a model using the **@model directive**. **Use** the model **with** **@Model**.

Only one **@model** can be used in view.

Same and different models can be considered for view and viewmodel. But its recommended to use different models for flexibility.

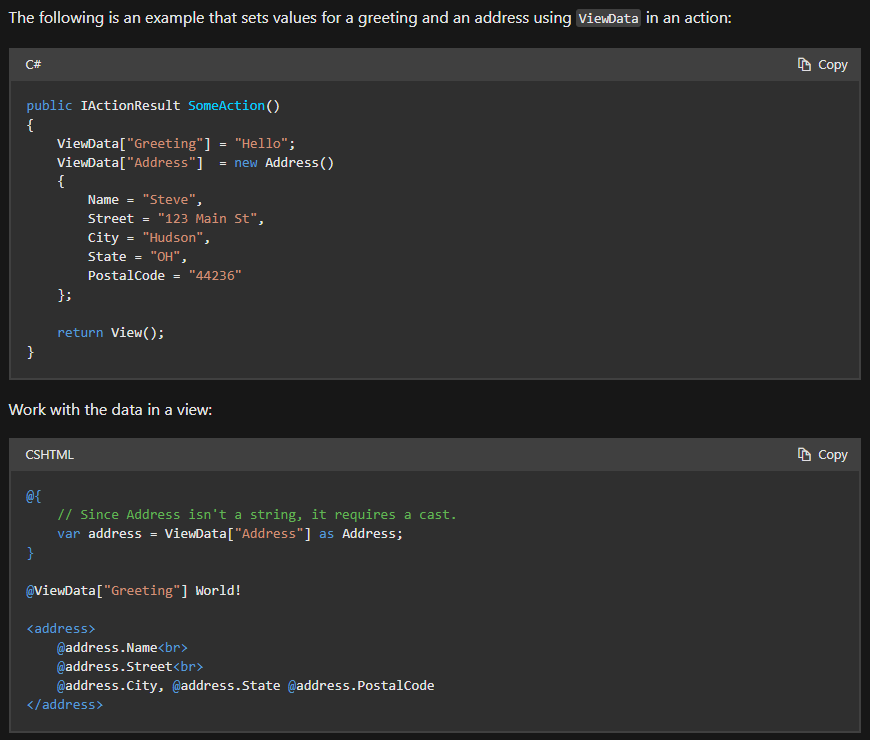
***SS Explanation: Address is a viewmodel. Passed as a parameter for View from contact controller.***



#### Weakly typed

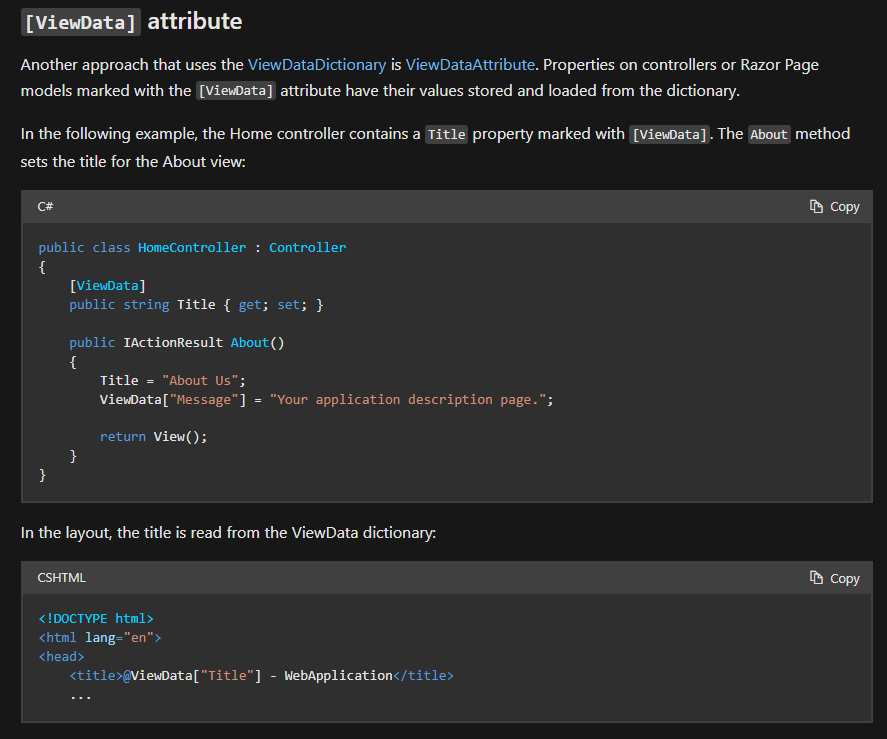
The **ViewData** property is a **dictionary** of weakly typed objects. The **ViewBag** property is a **wrapper around ViewData** that provides dynamic properties for the underlying ViewData collection.

Both are dynamically resolved at runtime.

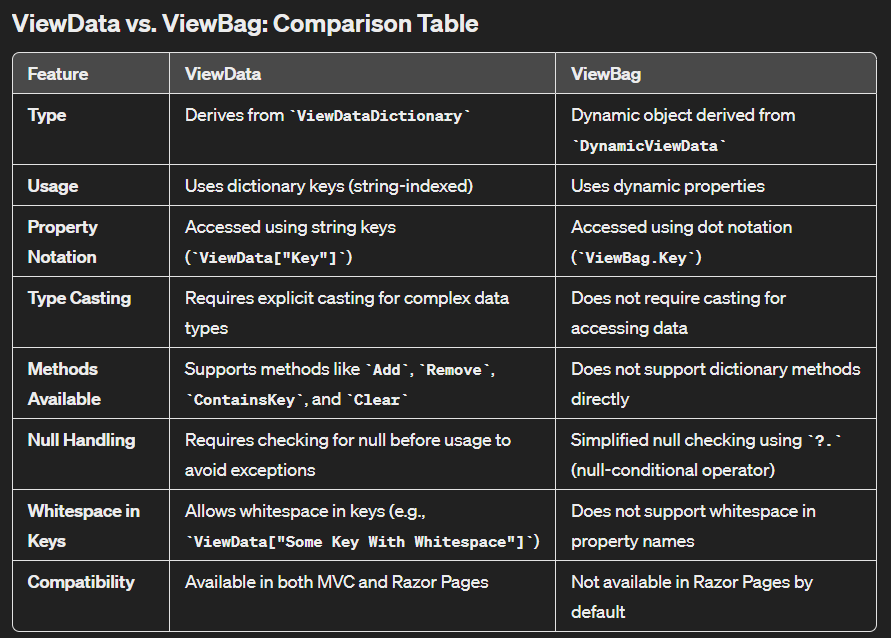


#### Viewdata attribute?

Add this attribute in controller and use it in view.



#### View data vs view bag



### Razor pages

## References

# C#

### Async await?

### TAP - Task based asynchronous programming?

### Assembly info file?

### Process vs thread vs task?

### Message queues

### LINQ

Language Integrated Query. Strongly typed syntax. One language for many sources.

Ways to implement: Query syntax and fluent syntax.

### LINQ Keywords with samples

### IEnumerable and IQueryable

# Dotnet core

## Features?

## Deferred execution?

# Web

## CDN?

## Rest Architecture

TODO ::: Update the six constraints

## Serverless architecture?

## What is SSL?

## Public key and private key

<https://www.youtube.com/watch?v=0ctat6RBrFo&t=343s>

## Symmetric and Asymmetric cryptography

## Http vs Https?

## Encryption algorithms

## Web sockets?

# Database

## Versions of SQL server

2022 is the latest version



## Indexing?

### Various indexing strategies?

## Normalization?

## Stored procedures?

## SQL vs No SQL

## Cursor?

## Constraints?

## Triggers?

## DBCC Commands

DBCC (Database Console Commands) in SQL Server are a set of commands that provide maintenance, validation, and other utility operations on a SQL Server database. These commands can be used for a variety of tasks such as checking database integrity, managing database storage, collecting and analyzing performance information, and more.

## Primary key vs unique key

# React

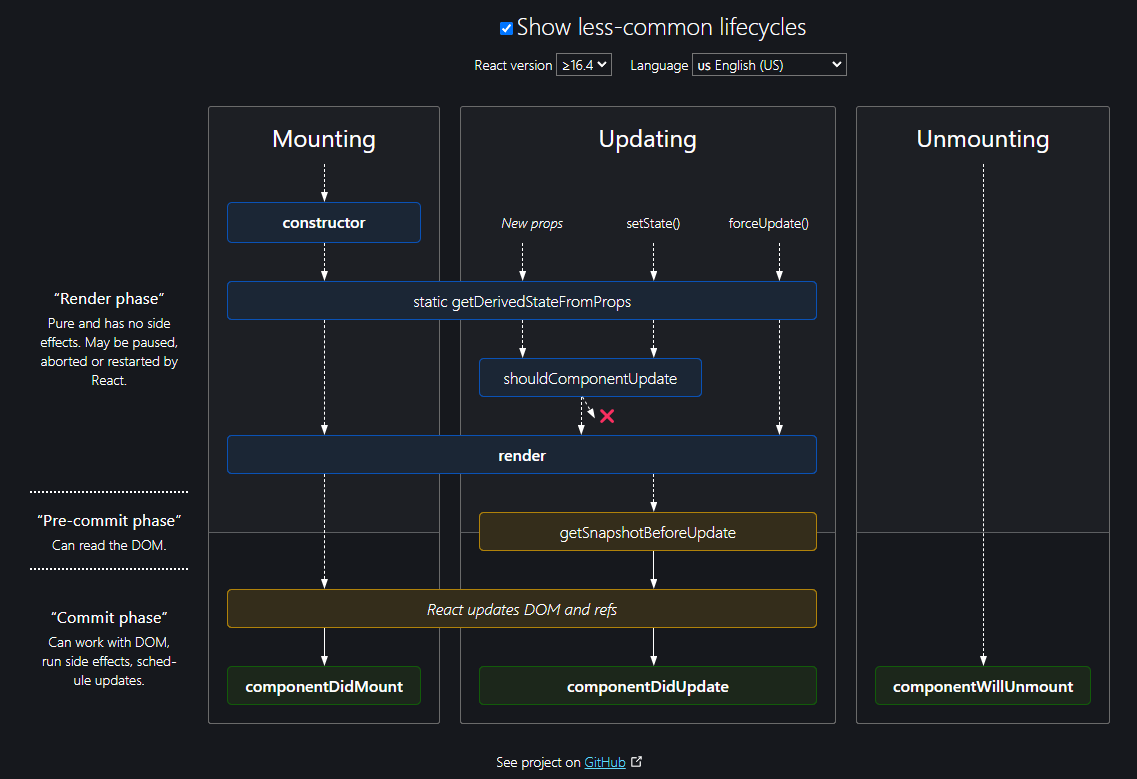
## Features of react?

## What is virtual DOM?

## Class vs functional component?

## Component lifecycle methods

Reference: https://projects.wojtekmaj.pl/react-lifecycle-methods-diagram/



### Constructor

### getDerivedStateFromProps

### Render

### ComponentDidMount

## Export vs export default

## Reconciliation?

Describes the process by which React updates the DOM based on changes in the component's state or props. The goal of reconciliation is to determine if and how the user interface should be updated in response to data changes.

Key Concepts in Reconciliation

1. **Elements of Different Types:**

If the elements have different types (e.g., from a <button> to an <a>), React will tear down the old tree and build the new tree from scratch, starting at that element.

1. **Elements of the Same Type:**

When comparing two elements of the same type, React keeps the underlying DOM node and only updates the changed attributes or properties. For instance, if the className of an element changes, React will only update the className attribute on the existing DOM node.

1. **Lists and Keys:**

When rendering lists, React uses keys to identify elements. Keys should be stable, predictable, and unique. By providing a unique key for each element, React can re-order, re-create, or remove elements efficiently during the reconciliation process.

## Diffing?

## Fiber tree?

## Hooks? Various hooks?

## Explain Usestate hook

## Explain Useeffect hook

## Explain Usecontext hook

## Write a custom hook

## Unmounting vs Re-rendering

# Javascript

## Closure

## Call, bind and apply

## Debounce and throttling

## References

https://github.com/sudheerj