MVC – Model

.NET CORE

*The **Model** in an MVC application represents the state of the application. Business logic should be encapsulated in the **model**, along with any implementation logic for persisting the state of the application.*

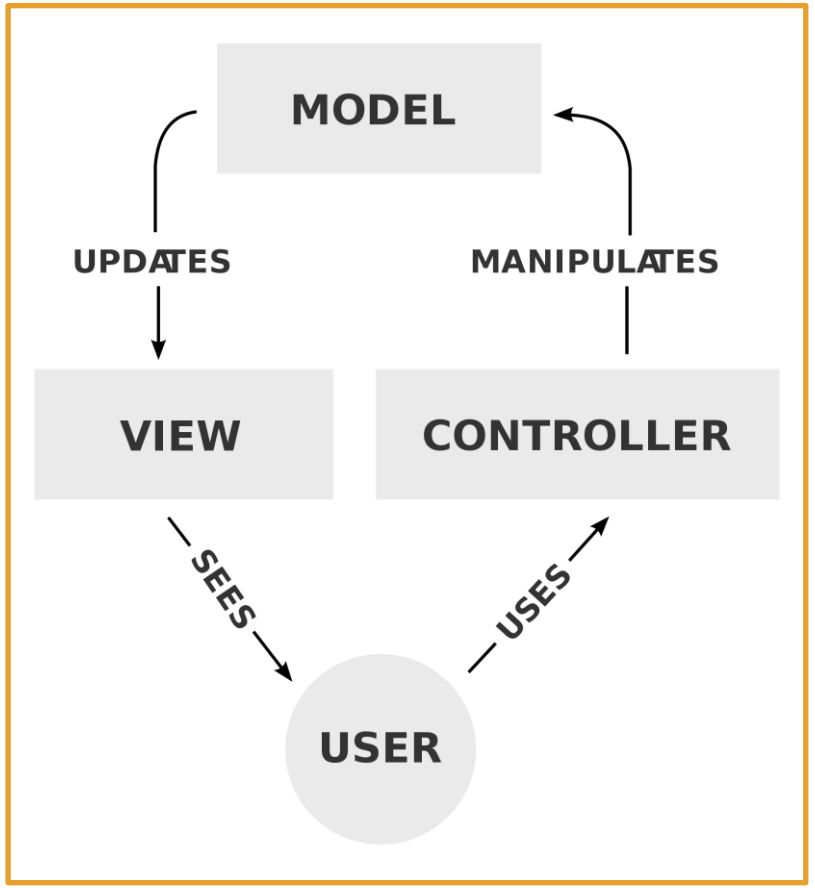
https://docs.microsoft.com/en-us/aspnet/core/mvc/overview?view=aspnetcore-3.1#model-binding

Model – Overview

https://docs.microsoft.com/en-us/aspnet/core/mvc/overview?view=aspnetcore-3.1#model-responsibilities
https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller

The ***model*** is the central component of the MVC pattern.

It is the application's dynamic data structure. The ***model*** receives user input from the ***view*** through the ***controller***. The model directly manages the data, logic and rules of the application.



Models – Examples

https://www.tutorialsteacher.com/mvc/mvc-model

A ***Model*** is a class that holds data in ***public properties*** that the program uses as a template for entities in the DB and, through ***model binding***, are connected to user input pages (***views***), ***controllers***, and the DataBase.

All ***Model*** classes reside in the ***Model*** folder in MVC folder structure.

```
namespace MVC_BasicTutorials.Models
{
    public class Student
    {
        public int StudentId { get; set; }
        public string StudentName { get; set;  }
        public int Age { get; set;  }
    }
}
```

Binding

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

Controllers work with data that comes from HTTP requests. The HTTP request could provide a record key or posted form fields may provide values for the properties of the ***model***. Writing code to retrieve each of these values and convert them from strings to .NET types would be tedious and error-prone. ***Model binding*** automates this process.

The ***model binding*** system:

- Retrieves data from various sources such as route data, form fields, and query strings.
- Provides the data to ***controllers*** as method ***parameters*** and public ***properties***.
- Converts ***string*** data to .NET ***types***.
- Updates ***properties*** of complex ***types***.

```
[HttpGet("{id}")]
public ActionResult<Pet> GetById(int id, bool dogsOnly)
```

Model Binding –

Examplehttps://docs.microsoft.com/en-us/aspnet/core/mvc/models/model-binding?view=aspnetcore-3.1#example

---

If your app receives the URL, http://myapp.com/api/GetByID/2?DogsOnly=true, the following steps happen on the action method,

```
[HttpGet("{id}")]
public ActionResult<Pet> GetById(int id, bool dogsOnly)
```

Model binding goes through the following steps after the routing system selects the **action method**:

1. Finds the first parameter of GetByID, an integer named id.
2. Looks through the available sources in the HTTP request and finds id = "2" in route data.
3. Converts the string "2" into integer 2.
4. Finds the next parameter of GetByID, a boolean named dogsOnly.
5. Looks through the sources and finds "DogsOnly=true" in the query string. Name matching is not case-sensitive.
6. Converts the string "true" into boolean true.
7. Calls the GetById method, passing in 2 for the id parameter, and true for the dogsOnly parameter.

Model Binding Targets

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

---

Model binding tries to find values for 3 kinds of targets:

- Parameters of the controller action method that a request is routed to.
- Parameters of the Razor Pages handler method that a request is routed to.
- Public properties of a controller or PageModel class, if specified by attributes.

The ***[BindProperty]*** attribute can be applied to a public ***property*** of a ***controller*** or PageModel class to cause ***model binding*** to target that ***property***.

```
[BindProperties(SupportsGet = true)]
public class CreateModel : InstructorsPageModel
{
    public Instructor Instructor { get; set; }
```

Apply [BindProperties] to a ***controller class*** to tell ***model binding*** to target all public ***properties*** of the class:

```
public class EditModel : InstructorsPageModel
{
    [BindProperty]
    public Instructor Instructor { get; set; }
```

Apply ***[BindProperty]*** to a public ***property*** of a ***controller*** to cause ***model binding*** to target that ***property***.

Attributes for Complex Data Types

https://docs.microsoft.com/en-us/aspnet/core/mvc/models/model-binding?view=aspnetcore-3.1#attributes-for-complex-type-targets

There are various different Annotation attributes for controlling ***model binding*** of complex ***types***. These attributes affect ***model binding*** when posted form data is the source of values.

A ***complex type*** must have a public default constructor and public writable properties to bind.

For binding to a parameter, the prefix is the parameter name. Some attributes have a Prefix property that lets you override the default usage of parameter or property name.

Attribute Name	Description
[BindRequired]	Applied only to model properties. Causes ***model binding*** to add a model state error if binding cannot occur for a model's property.
[BindNever]	Can only be applied to model properties. Prevents model binding from setting a model's property.
[Bind]	Can be applied to a class or a method parameter. Specifies which properties of a model should be included in model binding.

ModelState

https://docs.microsoft.com/en-us/aspnet/core/mvc/models/model-binding?view=aspnetcore-3.1#type-conversion-errors
https://docs.microsoft.com/en-us/dotnet/api/system.web.mvc.modelstate?view=aspnet-mvc-5.2

The ***ModelState Class*** encapsulates the state of ***model binding*** to a ***property*** of an ***action-method*** argument or to the argument itself.

If a source is found but can't be converted into the target ***type***, ***model state*** is flagged as invalid. The target parameter or ***property*** is set to null or a default value. In an ***API controller*** that has the ***[ApiController]*** attribute, invalid ***model state*** results in an automatic HTTP 400 response.

Otherwise, ***ModelState*** can be checked manually.

```
public IActionResult OnPost()
{
    if (!ModelState.IsValid)
    {
        return Page();
    }
```

Model Validation

https://docs.microsoft.com/en-us/aspnet/core/mvc/models/validation?view=aspnetcore-3.1

Both ***model binding*** and ***model validation*** occur before the execution of a ***controller action***. For web apps, if the controller doesn't have the ***[ApiController]*** attribute, it's the app's responsibility to inspect ***ModelState.IsValid*** and react appropriately. Web apps typically redisplay the page with an error message.

Validation attributes let you specify validation rules for model properties. Attributes are placed on the properties of the model.

```
public async Task<IActionResult> OnPostAsync()
{
    if (!ModelState.IsValid)
    {
        return Page();
    }

    _context.Movies.Add(Movie);
    await _context.SaveChangesAsync();

    return RedirectToPage("./Index");
}
```

```
[Required]
[StringLength(100)]
public string Title { get; set; }

[ClassicMovie(1960)]
[DataType(DataType.Date)]
[Display(Name = "Release Date")]
public DateTime ReleaseDate { get; set; }

[Required]
[StringLength(1000)]
public string Description { get; set; }

[Range(0, 999.99)]
public decimal Price { get; set; }
```

Built-in Attributes

https://docs.microsoft.com/en-us/aspnet/core/mvc/models/validation?view=aspnetcore-3.1#validation-attributes
https://docs.microsoft.com/en-us/dotnet/api/system.componentmodel.dataannotations?view=netcore-3.1

Attribute	Useage
[CreditCard]:	Validates that the property has a credit card format.
[Compare]	Validates that two properties in a model match.
[EmailAddress]	Validates that the property has an email format.
[Phone]	Validates that the property has a telephone number format.
[Range]	Validates that the property value falls within a specified range.
[RegularExpression]	Validates that the property value matches a specified regular expression.
[Required]	Validates that the field is not null. See [Required] attribute for details about this attribute's behavior.
[StringLength]	Validates that a string property value doesn't exceed a specified length limit.
[Url]	Validates that the property has a URL format.
[Remote]	Validates input on the client by calling an action method on the server. See [Remote] attribute for details about this attribute's behavior.

Passing Data to Views - ViewModel

https://docs.microsoft.com/en-us/aspnet/core/mvc/views/overview?view=aspnetcore-3.1#passing-data-to-views

---

You can pass Strongly-Typed data and Weakly-Typed data to ***views***. ***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***.

Strongly-typed - Viewmodel

Specify a ***model*** (aka, viewmodel) type in the view and pass it from the action method.
This allows the view to have ***strong type checking(***and ***Intellisense!)***. Strong typing (or strongly typed) means every variable and constant has an explicitly defined type (string, int, or DateTime). The validity of types used in a view is checked at compile time.

```
public IActionResult Contact()
{
    ViewData["Message"] = "Your contact page.";

    var viewModel = new Address()
    {
        Name = "Microsoft",
        Street = "One Microsoft Way",
        City = "Redmond",
        State = "WA",
        PostalCode = "98052-6399"
    };

    return View(viewModel);
}
```

Passing Data to Views
ViewData and ViewBag

https://docs.microsoft.com/en-us/aspnet/core/mvc/views/overview?view=aspnetcore-3.1#weakly-typed-data-viewdata-viewdata-attribute-and-viewbag

Weakly typed ViewData and ViewBag

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*.

```
public IActionResult SomeAction()
{
    ViewData["Greeting"] = "Hello";
    ViewData["Address"]  = new Address()
    {
        Name = "Steve",
        Street = "123 Main St",
        City = "Hudson",
        State = "OH",
        PostalCode = "44236"
    };

    return View();
}
```

Passing Data to Views
ViewData and ViewBag

https://docs.microsoft.com/en-us/aspnet/core/mvc/views/overview?view=aspnetcore-3.1#weakly-typed-data-viewdata-viewdata-attribute-and-viewbag

- ***ViewData*** is a Dictionary.
- ***ViewBag*** is a wrapper around ***ViewData*** that provides dynamic properties for the underlying ***ViewData*** collection.
- Key lookups are case-insensitive for both ***ViewData*** and ***ViewBag***.

ViewData and ***ViewBag*** don't offer compile-time type checking and are more error-prone than using a ***viewmodel***. Some developers prefer to minimally or never use ***ViewData*** and ***ViewBag***.

Weakly typed – ViewData, ViewDataAttribute, and ViewBag

Weakly typed 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.

Passing data between a ...	Example
Controller and a view	Populating a dropdown list with data.
View and a layout view	Setting the **<title>** element content in the layout view from a view file.
Partial view and a view	A widget that displays data based on the webpage that the user requested.

ViewData and ViewBag Differences

https://docs.microsoft.com/en-us/aspnet/core/mvc/views/overview?view=aspnetcore-3.1#weakly-typed-data-viewdata-viewdata-attribute-and-viewbag

ViewData()	ViewBag()
• Derives from ***ViewDataDictionary***, so it has dictionary properties that can be useful, such as ***ContainsKey***, ***Add***, ***Remove***, and ***Clear***. • Keys in the dictionary are strings, so whitespace is allowed. Example: ***ViewData["Some Key With Whitespace"]*** • Any type other than a string must be ***cast*** in the view to use ***ViewData***.	• Derives from ***DynamicViewData***, so it allows the creation of dynamic properties using dot notation (@ViewBag.SomeKey = <value or object>), and no ***casting*** is required. The syntax of ***ViewBag*** makes it quicker to add to ***controllers*** and ***views***. • Simpler to check for null values. Example: @ViewBag.Person?.Name • ViewBag isn't available in the Razor Pages.