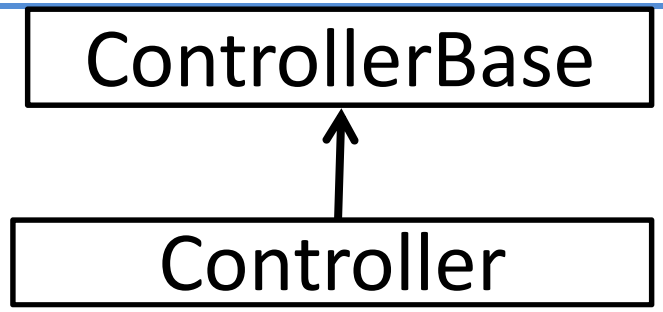
The Controller

- Controllers take requests from the user through action methods
- Controoler also is responsible for giving the output after processing back to the user.
- Controller must not do any logical processing.
- Controller must call model or an external service for processing.

Controllers



- Under the Microsoft.AspNetCore.Mvc namespace there are 2 classes
  1. Controller
  2. ControllerBase.
- ASP.NET Core Web Application controllers inherit from the Controller class,
- while ASP.NET Core WebAPI controller inherits from the ControllerBase class

Controllers

- Controller will contain one or more functions known as actions

Actions

- Actions are methods on a controller.
- Action should return any one of the following type as output.
  - IActionResult
  - Task<IActionResult>
  - or a class that implements IActionResult, such as ActionResult or the ViewResult.

ViewResult

- ViewResults and PartialView results are ActionResults that are returned from an action method
- ViewResult return type is to return a full HTML page as output
- Whereas PartialView is to return HTML snippet, which could be inserted to another views

View

View

- The View is the user interface of the application.
- Views are built using HTML, JavaScript, CSS, and Razor code.
- View optionally have a base layout view
- The View should be as lightweight as possible
- *Don't write any application logic inside the view*
- *View may contain commands for displaying result to user*

View

- View may contain HTML along with some Server Side commands
- The server side commands inside the view are processed by a **view engine**.
- There are multiple view engines that could be used.
- The default view engine is called **Razor**, and it uses the @ symbol to indicate server-side code.

Model

Model

- In ASP.NET Core MVC, the M stands for model.
- Models represent the data required to respond to a request.

The Model

- The Model is the data of your application.
- The data is typically represented by Plain Old CLR Objects (POCOs).

Note :- View models - one or more models that are shaped specifically for the consumer of the data.

Difference between ViewBag and ViewData

- ViewData will do auto boxing while storing the values
- ViewBag will not do auto boxing.
- ViewBag will keep the values in its original type.

Model Binding

Model Binding

- Model binding is the process where ASP.NET Core takes the name/value pairs available in the request and attempts to assign them to action parameters
- Implicit conversions are executed where applicable, such as setting a string property using an int value in the name/value pairs.
- If conversion doesn't succeed, that property is flagged in the ModelState as an error.

Model Binding

- Model Binding is the process in which data submitted as name-value pairs in an http request is mapped to Action's parameters
- Data from various sources such as route data, form fields, and query strings are mapped in ths step.
- When the action parameter is reference type then, the reference types must have a public default constructor, and the properties to be bound must be public and writable.

Using a Model to pass information to our View

- Controller action methods which return an ActionResult can pass a model object to the view.
- This allows a Controller to cleanly package up all the information needed to generate a response, and then pass this information off to a View template to use to generate the appropriate HTML response.

Model Binding

- Model binding will be performed in the following order.
  - Form values from an HTTP Post method
  - Route values provided through ASP.NET Core routing
  - Query string values

The ModelState Dictionary

The ModelState Dictionary

- The ModelState dictionary contains an entry for every item being bound through the model binder and items for the entire model itself.
- If an error occurs during model binding, the binding engine adds the errors to the dictionary entry for the property and sets the ModelState.IsValid = false.
- If all matched properties are successfully assigned, the binding engine sets ModelState.IsValid = true.

Model Validation

Model Validation

- Model validation occurs immediately after model binding
- Model validation is done to validate model based on business rules for example missing required fields, strings that exceed the maximum allowed length etc
- When all validation rule are followed ModelState.IsValid will be true and will be false otherwise.

Model Validation Attributes

Compare	Validates the two properties in a model match
EmailAddress	Validates the property has a valid email format
Url	Validates the property has a valid URL format
Range	Validates the property falls within a specified range
RegularExpression	Validates the property matches a specified regular expression
Required	Validates the property has a value
StringLength	Validates the property doesn’t exceed a maximum length