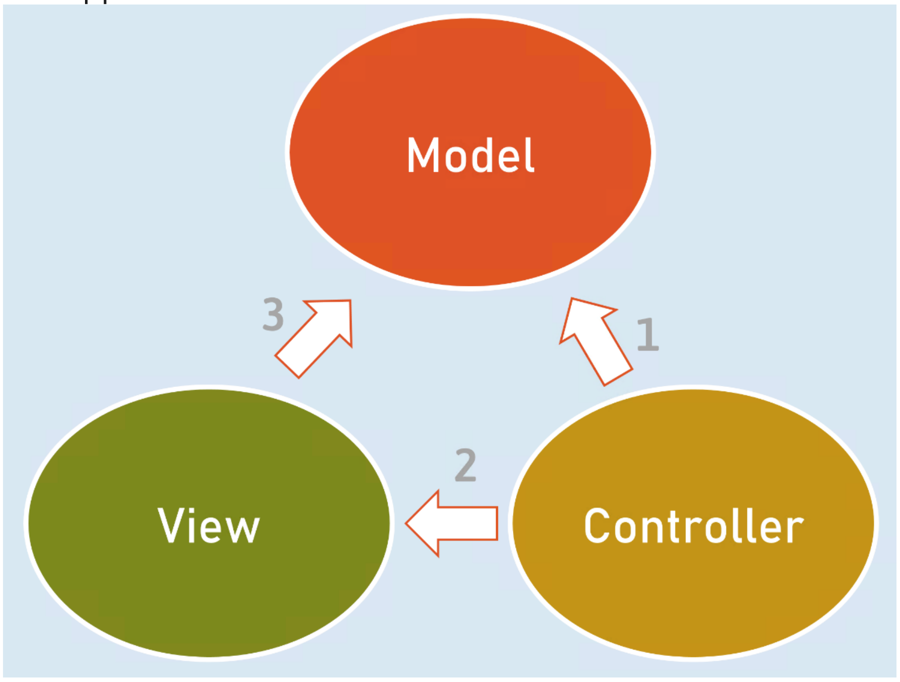
Instructor example

Web University by Harsha Vardhan

What is the MVC pattern?

The Model-View-Controller (MVC) is an architectural pattern that separates an application into three main logical components: the model, the view, and the controller. It’s different to note that this pattern is unrelated to the layered pattern we saw earlier. MVC pattern operates on the software side, while the layered pattern dictates how and where we place our database and application servers.



In an application that follows the MVC pattern, each component has its role well specified. For example, model classes only hold the data and the business logic. They don't deal with HTTP requests. Views only display information. The controllers handle and respond to user input and decide which model to pass to which view. This is known as the separation of responsibility or separation of concerns. It makes an application easy to develop and maintain over time as it grows in complexity.

Though Model-View-Controller is one of the oldest and most prominent patterns, alternate patterns have emerged over the years. Some popular patterns include MVVM (Model-View-ViewModel), MVP (Model-View-Presenter) and MVA (Model-View-Adapter).

Explain the role of the various components of the MVC pattern?

Model: Represents all the data and business logic that the user works within a web application. In ASP.NET core, the model is represented by C# classes that hold the data and the related logic that operates on that data. The 'Models' directory stores the model classes. You can also write POCO (Plain-Old-CLR-Object) classes only for storing the data and write business logic in a separate model class (a.k.a ‘Services’).

View: Represents all the UI logic of the application. In a web application, it represents the HTML that's sent to the user and displayed in the browser.

One important thing to remember is that all HTML code is not static or hard-coded. The HTML code in view can be generated dynamically using a model's data.

Controller: Acts as an interface between Model and View. It processes the business logic and incoming requests, manipulates data using the Model, and interacts with the Views to render the final output.

In ASP.NET, these are C# classes that form the glue between a model and a view. Controllers have action methods that act as middleware that execute upon receiving a HTTP request from the browser, then retrieve the data from model and pass it to the view to dynamically render a response. The controllers can be present in any folder – the common name of the folder is "Controllers".

Explain the differences between ViewData and ViewBag

1:

ViewData is Key-Value paired Dictionary collection.

ViewBag is internally "object" type.

2:

ViewData is a property of "Microsoft.AspNetCore.Mvc.Controller" class.

ViewBag is "dynamic" property of "Microsoft.AspNetCore.Mvc.Controller" class.

3:

ViewBag internally uses ViewData. So you can set / get data of ViewData using ViewBag.

ViewBag is the syntactic sugar to easily access the ViewData.

4:

Type Conversion code is required while reading values from ViewData.

Type conversion is not required while reading values from ViewBag.

5:

The lifetime of both ViewData and ViewBag is per-request. They will be deleted automatically at the end of the request processing. When a new request is received, a new ViewData will be created.

Explain strongly-typed views.

Strongly-typed views are tightly bound to a model.

The strongly-typed view can receive model object (of specific model class) from the controller.

The controller can supply model object by using the return View(model) method, at the end of action method.

You can use the strongly-typed tag helpers (such as asp-for) in strongly-typed views.

You will create a strongly typed view by mentioning the model class by using @model directive at the top of the view.

@model ModelClassName