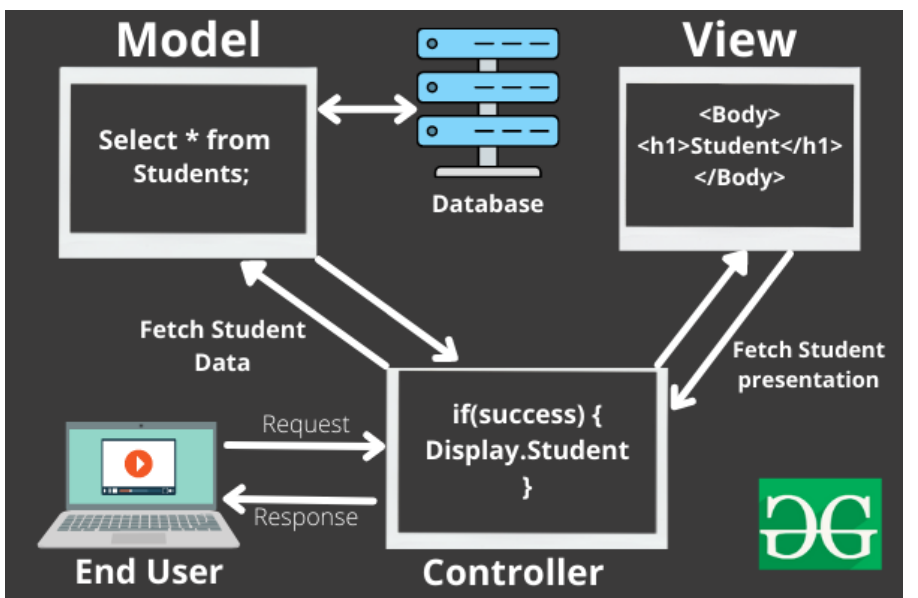
**Dot Net Document:**

**MVC Framework Introduction**

The [**Model-View-Controller (MVC)**](https://www.geeksforgeeks.org/mvc-design-pattern/) framework is an architectural/design pattern that separates an application into three main logical components **Model**, **View**, and **Controller**. Each architectural component is built to handle specific development aspects of an application



**Features of MVC :**

* It provides a **clear separation** of business logic, Ul logic, and input logic.
* It supports **Test Driven Development (TDD).**
* It offers full control over your HTML and URLs which makes it easy to design web application architecture.

**Disadvantages of MVC:**

* Increased complexity and Inefficiency of data
* It is not suitable for building small applications.

**MVC Fundamentals:**

// Add services to the container.

builder.Services.AddRazorPages();

builder.Services.AddControllersWithViews();

var app = builder.Build();

// Configure the HTTP request pipeline.

if (!app.Environment.IsDevelopment())

{

app.UseExceptionHandler("/Error");

app.UseHsts();

}

app.UseHttpsRedirection();

app.UseStaticFiles();

app.UseAuthorization();

app.MapDefaultControllerRoute();

app.MapRazorPages();

**Controller:**

The controller is the component that enables the interconnection between the views and the model so it acts as an intermediary.  It process all the business logic and incoming requests, manipulate data using the **Model**component and interact with the **View**to render the final output.

**View:**

The **View**component is used for all the UI logic of the application. It generates a user interface for the user. Views are created by the data which is collected by the model component but these data aren’t taken directly but through the controller.

**Model:**

The **Model**component corresponds to all the data-related logic that the user works with. This can represent either the data that is being transferred between the View and Controller components or any other business logic-related data.

**Razer View:**

Razor View engine is a markup syntax which helps us to write HTML and server-side code in web pages using C# or VB.NET.

Razor is a templating engine and ASP.NET MVC has implemented a view engine which allows us to use Razor inside of an MVC application to produce HTML. However, Razor does not have any ties with ASP.NET MVC.

**Explanation about IIS Express**

@{

var price = 101;

}

@{

if(price == 100)

{

<p>It's hundred</p>

}

else

{

<p>It's not hundred</p>

}

}

@model Employee

<h2>Employee Detail:</h2>

<ul>

<li>Student Id: @Model.EmployeeId</li>

<li>Student Name: @Model. EmployeeName</li>

<li>Age: @Model.Age</li>

</ul>

**Routing in ASP.NET Core MVC**

Routing is the process through which the application matches an incoming URL path and executes the corresponding action methods. ASP.NET Core MVC uses a routing middleware to match the URLs of incoming requests and map them to specific action methods.

There are two types of routing for action methods:

* [Conventional Routing](https://code-maze.com/routing-asp-net-core-mvc/#conventionalrouting)
* [Attribute Routing](https://code-maze.com/routing-asp-net-core-mvc/#attributerouting)

**Conventional Routing**

Configure routing in Configure method in startup.cs class

app.UseEndpoints(endpoints =>

{

endpoints.MapControllerRoute(

name: "default",

pattern: "{controller=Home}/{action=Index}/{id?}");

endpoints.MapControllerRoute(

name : "employee",

pattern: "EmployeeDetails",

defaults: new { controller = "Employee", action = "Index" });

});

**Attribute Routing**

By placing a route on the controller or the action method, we can make use of the Attribute Routing feature.

Let’s modify the Configure() method in the startup.cs class and remove the default routes that we had defined earlier.

[Route("[controller]/[action]")]

[Route("[controller]")]

public class TestController : Controller

{

[Route("")] // Matches 'Test'

[Route("Index")] // Matches 'Test/Index'

public IActionResult Index()

}