MVC Core

* Asp .net Core application starts as a console application.
* Main method in Program.cs is the entry point of the app execution. It Configures the Asp.net Core and that’s when asp.net Core application becomes Web application.

Q. Which file is first used during execution? – Program.cs (Main method)

Q. What is CreateDefaultBuilder in Progam.cs?

– Sets up the web server which hosts our application.

-Loading the host and application configuration from various configuration sources .

* There are 2 types of web server hosting in Asp .Net
  + In-Process (By default)
  + Out-Of-Process []
* We use IIS in Production and IISExpress(light-weight) in Development.

**In Process Hosting:**

**-**Only 1 Web server.

**-**InProcess hosting is better than out-of-Process hosting from performance standpoint.

**-**Direct connection from internet to IIS Server

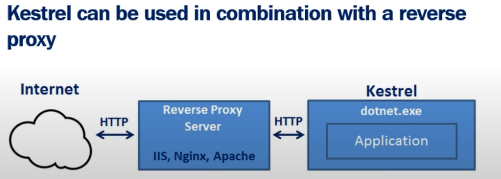
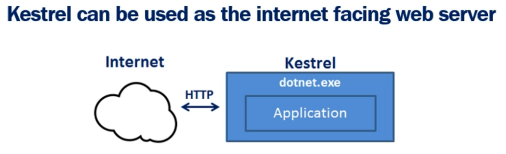


**Out Of Process Hosting:**

-Has 2 Web servers:

-Internal (Kestrel)

- External (IIS, Nginx, Apache).



Q. What is Kestrel? – Cross Platform Web server for asp.net.

Q Why do we need Reverse Proxy Servers?

– Provides Additional configurational capabilities.

Q. What is a Middleware?

-A piece of software which handles httpRequest or httpResponse.

-We can have Middle ware for Error handling, File reading, user authentication, etc.

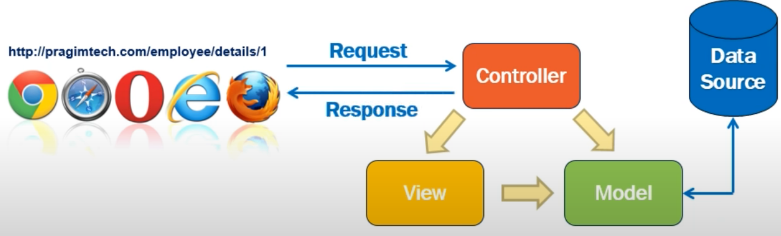
-Middlewares are executed in the order they are added.



-Logging, Static Files and MVC are Middle Ware

-Middleware short-circuiting is when the Middleware pipeline is Partially executed due to absence of some info in some middle ware which might be desire-able in some situations.

**Flow of Data**



Q. What is Dependency Injection? – A technique in which an object receives other objects that it depends on, called dependencies. Receiving Object is Client and passed-in object is a Service.

Eg: public class ExampleC1{

Private IExampleService \_service;

Public ExampleC1(IExampleService service){

\_service = service;

}

}

Services.AddScoped<IExampleServiceRepository, ExampleServiceRepository>(); [Startup.cs]

Q. Need for Dependency Injection? Why not make a constructor for Example Service?

-Making a constructor instead will tightly couple the Classes and incase if we later on make a new implementation(ExampleServiceRepository) of the interface than we will have to change every created instance line. This would not matter is less object/instances are created but would take a long time if numerous objects/instances are created as we will have to manually change them all.

* There Are 3 different methods used to register a dependency in startup.cs
  + AddSingleton()
    - Creates Singleton service
    - An instance is created when first requested and that instance is used by all the subsequent request.
  + AddTransient()
    - Instance created each time it is requested.
  + AddScoped()
    - Created Once per request.
    - An instance created for a HttpRequest is used for all HttpRequest for the same Web Request.
* Three Ways to Pass data from Controller to View:
  + ViewData
  + ViewBag
  + Strongly Typed View
* **ViewData (**Loosely Typed**)**

-Syntax: ViewData[“name\_it\_here”] = model; [controller]

@ViewData[“name\_it\_here”] as MODEL\_HERE [Razor Page]

-Eg :

ViewData[“Title”] = “Employee Portal”;

<h1>@ViewData[“Title”]<h1>

Draw back is that it is a loosely typed view meaning the data we are storing is not kown by the view. In case the name/key of our ViewData is mis-typed there will be no compile time error.

* **ViewBag (**Loosely Typed**)**

-Syntax: ViewBag.name\_your\_bag; [controller]

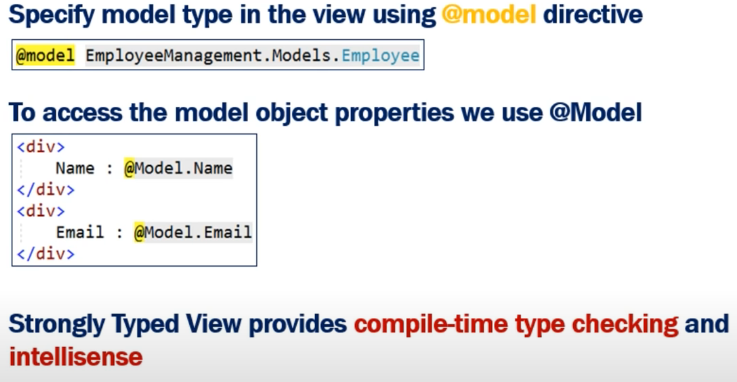
@ViewBag.name\_your\_bag [Razor Page]

-Eg :

ViewBag.Title = “Employee Portal”;

<h1>@ViewBag.Title<h1>

* **Strongly Typed**



* **ViewModel**

Q. What is a View Model? – We create a view model when the view does not need all the data a model object contains

Q. Why do we use View Model? – So as to not expose all the model Object data which is required by the view or to NOT expose sensitive irrelevant data like ID and such.

* **Layout View**

Layout view is the common view which is same for every view

Eg: Top Navbar which is the same in every view

Q.Why do we use Layout view? – To avoid repetition of code for an element used in multiple views.

-We Place Layout View in the Shared folder as it does not belongs to a specific controller.

-We can have multiple Layout views.

Eg: 1 layout for admin user and 1 for customer.

* \_ViewStart

-Views Start is used for setting up the layout file for al the views

Eg: @{

Layout = \_LayoutView\_Name

}

-The layout can be overridden if we mention null or another file name in a specific view.

Eg: @{

Layout = null / SpecificLayoutView

}

-We can also Implement conditional checks in the \_ViewStart file to load views for Admin or User roles.

-Code in \_ViewStart file is executed before any individual view

* **ViewImports**

-Used to include the common namespaces.

Eg: @using namespace\_name\_here

-IT is placed int the Views Folder.

* **Routing**

-There are two types of routing

i. Conventional Routing

ii. Attribute Routing

-In Startup.cs

‘App.UseMvc();’ adds mvc support and eventually routing support

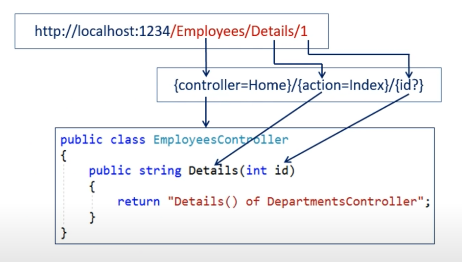
-Specifying routing template

App.UseMvc(routes =>{

Routes.MapRoute(“default”,”{controller=Home}/{action=Index}/{id?}”)

});

-Route (conventional)



Attribute Routing

-We can place the attribute route n the controller itself or individual actions methods

Eg:

[Route(“”)]

[Route(“Home”)]

[Route(“Home/Index”)]

Public IActionResult Index(){

}

-To specify a route parameter like ‘id’ we use curly braces

Eg:

[Route(“Home/Details/{id?}”)]

Public IActionResult Details(int id){

}

-For Common Part of route, like in this case ‘Home’ We can specify it at the controller

Eg:

[Route(“Home”)]

Public class HomeController

-To make it more dynamic we can use ‘[controller]’ and ‘[action]’ in place of controller and action method names, respectively.

* **Tag Helpers**

Manually generating Links-

[-@Url.Action(“action”,”Controller](mailto:-@Url.Action()”, new {route\_parameter = variable\_name})

[-@Html.AcitionLink(“View”,”action\_method”,”controller](mailto:-@Html.AcitionLink()”, new {route\_parameter = variale\_name})

Both can be used in button and anchor tags

Using Tag Helpers-



-They are used just so in case we change the default template for routing after the development is done, it doesn’t break the project.

* Entity Framework Core

-Needed for efficient data transfer.

-To fully enable the functionality, we need three packages

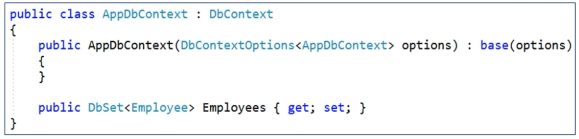
Microsoft.entityframeworkCore.Sql

Microsoft.entityframeworkCore.Relational

Microsoft.entityframeworkCore

-Relational is dependent on Core and Sql is dependent on Relational.

* **DbContext**



**-**The DbContext class includes a DbSet<TEntity> property for each in the model.

-We will use this DbSet property Employees to query and save instances of the employees class.

-The LINQ queries against the DbSet<TEntity> will be translated into queries against the underlying database.

-Add Dbcontext in StartUp.cs ConfigurationService like

*Services.AddDbContext(<AppDbContext/Filename here> (options =>*

*Options.UseSqlServer(ConnectionString));*

(in User secret or appsetting.json)

*ConnectionString:{*

*“EmployeeDBConnection”:”server=(localdb)*[*\\MSSQLLocalDB*](file:///\\MSSQLLocalDB)*; database=dbname;Trusted\_Connection=true”*

-There area two options, addDbContext and addDbContextPool

-addDbContext creates new instance everytime it is called.

-addDbContextPool checks in the Pool for already created instance and uses that if found. Hence in terms of performance this is more efficient.

* Identity Table