|  |
| --- |
| Resources |
| <https://docs.microsoft.com/en-us/aspnet/core/fundamentals/app-state?view=aspnetcore-6.0> |
| <https://www.youtube.com/watch?v=pXmMdmJUC0g> |
|  |

ASP.NET MVC

* MVC
  + Model View Controller
* Return Types
  + View result
  + Json Result
  + Content Result
  + Redirect Result
  + JavaScript Result
* Filters
  + Authorization Filter
  + Action filter
    - Different action filters
      * Output cache
      * Handle Error
      * Authorize
  + Result/Response filter
  + Exception filter

(order of execution Authorization Filter > Action Filter > Response Filter > Exception Filter)

* System.Web.MVC – framework where MVC assembly defined
* Benefits of MVC
  + Enable full control over the rendering of HTML
  + Separation of Concern (Soc)
  + Enables Test Driven Development Approach
  + Light weight
* Can we maintain session in MVC?
  + View data – controller to view (applicable in same controller)
  + Temp Data – can pass from one controller to another
  + View Bag – used to pass data from controller to view. view bag dynamic wrapper

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Maintain Data Between | ViewData/ViewBag | TempData | Hidden field | Session |
| Controller to Controller | No | Yes | No | Yes |
| Controller to View | YES | No | No | Yes |
| View to Controller | No | No | Yes | Yes |

* Keep v/s Peek in TempData
  + Once you read data from TempData in the current request it’s not available for subsequent requests. To available for subsequent requests, we need to keep the data.

@TempData[“SomeData”];

TempData.Keep(“SomeData”);

More shortcut way to achieving using Peek. (read and save data in one go)

String data = TempData.Peek(“”someData).ToString();

* Session Management in MVC?
  + Session management techniques helps to maintain state between HTTP requests.
  + As we know, HTTP is a stateless protocol call. Which means it will not remember about the user state. Once a request is handled and response is served. HTTP will not remember about that request. Which means, what the user is request and what the response we server.
  + In other words, every request is considered as a new request.
  + To manage the session between the state, will use session management.
* Various ways for state management. Remember each one has its own significant

|  |  |
| --- | --- |
| Cookies | Http Cookies, may include data stored using server-side app code |
| Session state | Http Cookie and service-side app code |
| ViewData / ViewBag / TempData | Http Cookie or session state |
| Query String | HTTP query string |
| Hidden fields | HTTP from fields |
| HttpContext.items | server-side app code |
| Cache | server-side app code |

**Session**

* Session stores the user data while browsing the web app
* Empty session isn’t persisting. Session must have at least one value set to persists the session across request.
  + Configure Session state
    - Step#1: first import Microsoft.AspNetCore.Session namespace in startup.cs
    - Step#2: we can use the middle wares AddSession() and UseSession() in ConfigurationService and configure method accordingly.
  + Order of middleware is import \*\* call UseSession() after UseRouting and before MapRazorPages() and MapDefaultControllerRoute().
* How to enable sessions in MVC Core? (call AddSession())

|  |
| --- |
| public void ConfigureServices(IServiceCollection services)  {  services.AddSession(); // also we can pass SessionObject as parameter  } |

|  |
| --- |
| Public void Configure(IApplciationBuilder app, IWebHostEnvironment env)  {  App.UseSession(); // also we can pass SessionObject as parameter  } |

* **Session Options:** Cookie, IdleTimeout, IOTimeOut

|  |
| --- |
| services.AddSession(options =>  {  options.Cookie.Name = ".Home.Session";  options.IdleTimeout = TimeSpan.FromSeconds(50);  options.IOTimeout = TimeSpan.FromMinutes(1);  options.Cookie.IsEssential = true;  }); |

* Saving session
  + HttpContext.Session.SetString(“SessionData”,MemberObj);
* Accessing session
  + MemberObj sessionData = HttpContext.Session.GetString(“SessionData”);
* Important segments of routing
  + Controller Name
  + Action Method Name
  + Parameters
* Different properties of MVC routes
  + Route Name
  + URL Pattern
  + Defaults
  + Constraints
* Razor: It’s a view engine where we can write both c#/VB.net code and HTML
* Navigating from One view to another view
  + Html.ActionLink(“Home”,”Go to Home”)
* ViewState
  + It’s a layout which is executed at the beginning and applied to all the views.
* View Model
  + It is a define as a plan class with different properties
  + It is used for binding a view that is strongly typed
  + It consists of various validation rules and data annotation
* Action Types
  + Get: Used to get data
  + Post: use to submit or persists data
* Can we map multiple URLs to the same action?
  + Yes, you just need to make two entries with different key names and specify the same controller and actions.
* ActionResult v/s ViewResult
  + **ActionResult** – if we want to return different types of view dynamically at runtime, then ActionResult is the best choice.
    - ActionResult can be used to exploit polymorphism and dynamism

Below example demonstrate based on flag (isHTML), it either return ViewResult or JsonResult.

* + **ViewResult** – it derived from ActionResult, which is capable of returning views.

|  |
| --- |
| Public ActionResult DynamicView()  {  If(isHTML)  Return View();  Else  Return Json();  } |

* How can we detect that mvc controller called by GET or POST method?
  + Request.HTTPMethod used to detect whether a method called with GET or POST

|  |
| --- |
| Public ActionResult DoSomething()  {  If (**Request.HTTPMethod** == “POST”)  Return View(“Page1”);  Else  Return View(“Page2”);  } |

* Bundling and Minification in MVC
  + **Bundling**: web projects always need multiple CSS and java script files on each request. Bundling help, us to combine multiple java script and css file into single entity thus minimizing multiple requests into a single request.
  + **Minification**: it is the process of compressing files without spaces and line break to minimize file size. (removes blank spaces, comments etc)
* **App\_Start > BundleConfig**.cs file to add script files
* Debugging in Bundling file
  + If we want to debug the bundling file, we need to set BundleTable.EnableOptimization = true; from bundleConfig.cs file.
* Areas in MVC
  + Areas help us to group functionalities into independent modules thus making the project more organized.
* Exception Handling in MVC
  + In the controller you can override **OnException** event and set the Result to the view, which we want to render on Exception occurs.
* Who we can handling multiple submit methods pointing to multiple actions methods in MVC?
  + We can achieve using AJAX
  + We can handle using two different HTML form for each button click
* Why do we use wwwroot folder in MVC?
  + We store the static content like jc, css, lib files
* Importance of appsettings.json
  + It will help us to store the configuration data of project
* How to read configuration data from appsetting.json?
  + To read the config data from appSetting.json we use the IConfiguration object (which is the part of Microsoft.Extensions.Configuration) which will injected by MVC CORE Dependency Injection.

**Case#1**

If the connection string has KEY:VALUE pair, like

“DbConnectionString” : “DataSource=Demo”

Public DemoController(Demo demo, IConfiguration configuration)

{

String conn = configuration[DbConnectionString];

}

**Case#2**

if connection string has an object, like

{

Environment: {

“Prodution”: ”No”,

“Test” : “Yes”,

“DEV” : “No”

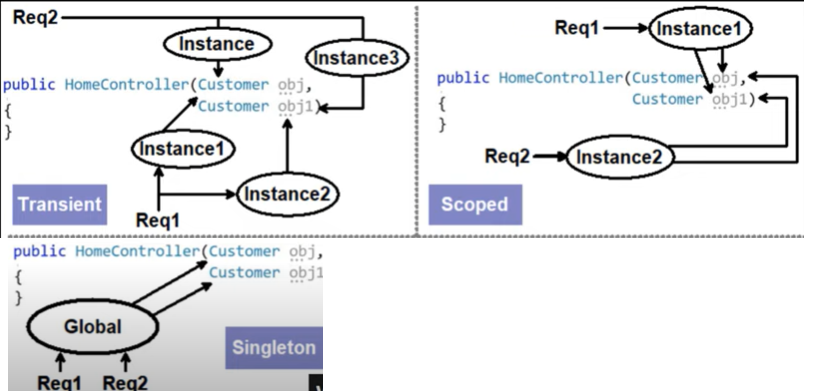
}

}

Will use colon (:) to access the key, like

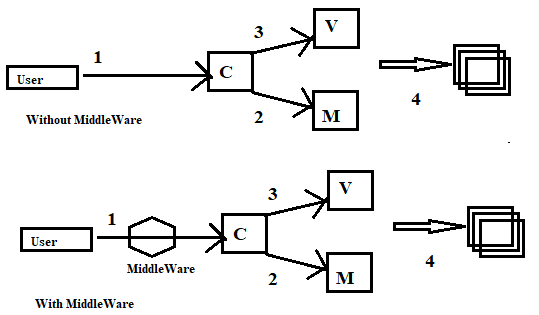
String environment = configuration[Environment:Production];

* Dependency Injection
  + Why dependency injection
    - To achieve inversion of control: Simple words we need to stop people for creating new object during programming.
  + How to achieve dependency injection
    - To implement DI, we need to use provide DI object in the ConfigurationServices method (startup.cs class) using **Scoped**, **Singleton**, or **Transient**
* **Scoped**
  + instantiates once per request made to the server
  + it will create **one instance** for every dependency injection object
* **Transient**
  + Instantiates every time service is injected
  + It will create a **new instance** for every dependency injection object
* **Singleton**
  + Instantiates once in an application lifetime
  + it is like a global object



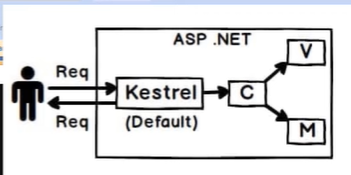
|  |
| --- |
| Startup.cs >  Public void ConfigurationService(IServiceCollection services)  {  Services.AddScoped<EmployeeDbContext>();  } |

* Middle ware
  + Middleware helps to execute the pre-processing logic before the request hits to the controller

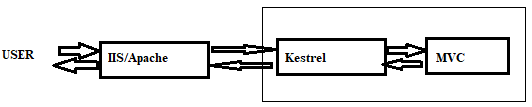


* How to add MiddleWare in project
  + Step#1: Add a middle ware class using the template (middleware.cs)
  + Step#2: Write the pre-processing logic in the **Invoke** (HttpContext context) method
  + Step#3: Plug the Middle Ware – go to the Startup.cs class, in the Config method plug the middle ware as
    - app.UseMiddleware<CustomAuthentication>();
* Configure v/s ConfigureServices methods in startup.cs
* Both methods are called at Runtime.
  + ConfigureService
    - Use this method to **add** services to the container
    - E.g) Dependency Injection objects
    - All .Add(s) go here
  + Configure
    - Use this method to configure the HTTP request pipeline
    - E.g) use to configure middleware, routing, authorization etc
    - All .use go here
* What is Kestrel Web Server?
  + Kestrel web server is an open-source web server which comes with ASP.NET MVC Core
  + Kestrel is a default web server with MVC Core
* Why we need Kestrel when we have Microsoft IIS web server?
  + Remember, ASP.Net MVC Core is designed to work on cross platform and IIS is specific to Microsoft windows.
* Does Kestrel replace the IIS Server?
  + No, IIS Server or Apache tomcat are long time tested and running servers. In production environment we follow the **Reverse Proxy** mechanism.
  + **Reverse Proxy**: It’s a mechanism where the first request hit to the IIS server and send to Kestrel, from kestrel to controller.
  + In this mechanism, Kestrel is just acting like a mediator between web application and IIS Server.

**Kestrel Web Server**



**Reverse Proxy**



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **MVC 2** | **MVC 3** | **MVC 4** | **MVC 5** | **MVC 6** |
| Client-side validations | Razor | ASP.NET WEB API | One asp.net | ASP.Net MVC & Web API merged into one |
| Templated helpers | Readymade project template | Refreshed and modernized default project templates | Attribute based routing | Dependency injection is in built and part of MVC |
| Areas | HTML 5 enabled templates | New mobile project template | Asp.net Identity | Side by Side – deploy the runtime and framework with your application |
| Asynchronous controllers | Support for multi view engines | Many new features to support mobile apps | Bootstrap in MVC template | Everything packaged with Nuget, including .Net runtime engine |
| HTML.ValidationSummary Helper Methods | Javascript and Ajax | Enhanced support for Asynchronous methods | Authentication Filters | New Json based project architecture |
| Default value attribute in action method parameter | Model validation improvements |  | Filter overrides | No need to recompile for every change. Just save and refresh the browser |
| Data annotations attributes |  |  |  | Compilation is done with new Roslyn real time compiler |
| Model validators providers |  |  |  | vNext is Open Source via the .NET foundation and it is taking public contributions |
| Templated helpers |  |  |  | nNext & Roslyn also run on MONO, on both MAC and linux. |
| Display model level errors |  |  |  |  |

* Different approaches to pass multiple view models to view (e.g View Models: FeedbackViewModel and CommentViewModel)

**Approach#1 View Model –** use a new View Model class as wrapper and bind above views as properties

|  |
| --- |
| Public class ManageViewModel  {  Public FeedbackViewModel feedback {get; set;}  Public CommentViewModel comment {get; set;}  } |

**Approach#2 ViewBag** **/ ViewData /TempDate**

Using View Bag, we can pass data from controller to view.

View bag scope is only during the current request

|  |
| --- |
| **Controller**  Public ActionResult GetManageData()  {  ViewBag.feedback = GetFeedbackModel();  ViewBag.comment = GetCommentmodel();  } |
| **View**  @ViewBag.Feedback.Description (you can access any property like this)  @ViewBag.Comment.Note |

**Approach#3 Using Session object**

|  |
| --- |
| Controller  Public ActionResult ManageDataViewModel  {  Session[“feedback”] = GetFeedbackModel();  Session[“comment”] = GetCommentModel();  } |
| **View**  @feedback.Description  @comment.note |

* **Caching** 
  + **In Memory caching**
    - It stored in the memory of a single server hosting the application. Basically, data will be cached within the application
  + **Distributed caching**
    - Distributed cache is a cache that is shared by one or more application and it is maintained as external services that is accessible for all the servers. so, distribution cache is external to the application
    - main advantage of distribution cached is, this is consistent across multiple servers.
    - We can implement distributed cache with **RedIs**
* **Redis:** it is an open source, in memory data structure store. It is the key – value based and no sql database.
* Cache in v/s Cache miss
  + Cache In – A cache hit occurs when data found in a cache
  + Cache Miss – it occurs when data not found in cache
* **In Memory Caching Implementation**
  + Step#1- use .AddMemberCache() (which is available in Microsoft.Extensions.Caching.Member namespac) in ConfigureService() method of statup.cs class
  + Step#2 – In the controller implement cache mechanism
    - Check if cache exists
    - Implement server calls
    - Setup cache options
    - Setup cache entries

Step#1

|  |
| --- |
| public void ConfigureServices(IServiceCollection services)  {  services.AddMemoryCache();  services.AddControllers();  } |

Step#2

|  |
| --- |
| public async Task<IActionResult> GetZipCodes()  {  var cacheKey = "zipcodes";  // check if cache exists  if(!\_memoryCache.TryGetValue(cacheKey, out List<Zipcode> zipCodeList))  {  // calling the server  // assuming we are get data from server  zipCodeList = new List<Zipcode>(); // in real, we need to get data from context object  // set up cache options  var cacheExpireOption = new MemoryCacheEntryOptions {  AbsoluteExpiration= DateTime.Now.AddMinutes(10),  Priority = CacheItemPriority.High,  SlidingExpiration = TimeSpan.FromSeconds(20)  };  // setting cache entries  \_memoryCache.Set(cacheKey, zipCodeList, cacheExpireOption);  }  return Ok(zipCodeList);  } |