2. ASP.NET Core Request Processing Pipeline

2.1 Middleware

- Middleware is a software component that sits between the web server and the application. It intercepts incoming requests and outgoing responses and allows you to modify them. Middleware can be used to perform a wide range of tasks such as authentication, logging, compression, and caching.
- Middleware works by using a pipeline model. When a request comes in, it is passed through a series of middleware components before it reaches the application. Each middleware component can modify the request before passing it on to the next component. Once the request has passed through all the middleware components, it reaches the application. Similarly, when a response is sent back to the client, it is passed through a series of middleware components before it is sent back to the client.
- There are two types of middleware in .NET Core : terminal and non-terminal middleware. Terminal middleware is the final middleware component in the pipeline. It is responsible for sending the response back to the client. Non-terminal middleware is any middleware component that is not the final component in the pipeline.
- Terminal Middleware
- The terminal middleware is responsible for sending the response back to the client. It is the final middleware component in the pipeline. Terminal middleware can be used to modify the outgoing response before it is sent back to the client.
- Some examples of terminal middleware include:
    - Static files middleware: This middleware is used to serve static files such as CSS, JavaScript, and images.
    - File server middleware: This middleware is used to serve files from a specified directory.
    - MVC middleware: This middleware is used to handle requests for MVC endpoints.
- Non-terminal Middleware
- Non-terminal middleware is any middleware component that is not the final component in the pipeline. Non-terminal middleware can be used to modify incoming requests and outgoing responses.

- Some examples of non-terminal middleware include:
    - Authentication middleware: This middleware is used to authenticate users.
    - Authorization middleware: This middleware is used to authorize users to access certain resources.
    - Response compression middleware: This middleware is used to compress the response before it is sent back to the client.
    - Request logging middleware: This middleware is used to log requests.
    - Routing middleware: This middleware is used to route requests to the appropriate endpoint.
- Custom Middleware
    - In addition to the built-in middleware components that are available in .NET Core, you can also create your own custom middleware components. Creating custom middleware allows you to add functionality to your application that is specific to your needs.
    - To create custom middleware, you need to create a class that implements the IMiddleware interface. The IMiddleware interface has a single method called InvokeAsync that is called when the middleware component is invoked.

2.2 Routing

- Routing is way of matching request URL with source like controller's action etc.
- There are two types of routing
    1. Conventional Routing :
       It is used for global routing purposes.
       Gives less flexibility/control compare to attruibute based routing.
       It is safe to use conventional routing since it creates less or no confussion or conflicts.
    2. Attribute-Based Routing :
       It is used for particular action method of controller.
       It gives more control over action methods compare to conventional routing.
- We can also use 'Routing constraints' to restrict data comes from URL.

2.3 Filters

- Filters allows us to run custom code before or after executing the action method. They provide ways to do common repetitive tasks on our action method. The filters are invoked on certain stages in the request processing pipeline.
- There are many built-in filters available and we can create custom filters as well. Filters help us to remove duplicate codes in our application.
- Every filter type is executed at a different stage in the filter pipeline.
- Following are the filter types :
    1. Authorization filters :

       The Authorization filters are executed first. This filter helps us to determine whether the user is authorized for the current request. It can short-circuit a pipeline if a user is unauthorized for the current request. We can also create custom authorization filter.
    2. Resource filters :

       The Resource filters handle the request after authorization. It can run the code before and after the rest of the filter is executed. This executes before the model binding happens. It can be used to implement caching.
    3. Action filters :

       The Action filters run the code immediately before and after the controller action method is called. It can be used to perform any action before or after execution of the controller action method. We can also manipulate the arguments passed into an action.
    4. Exception filters :

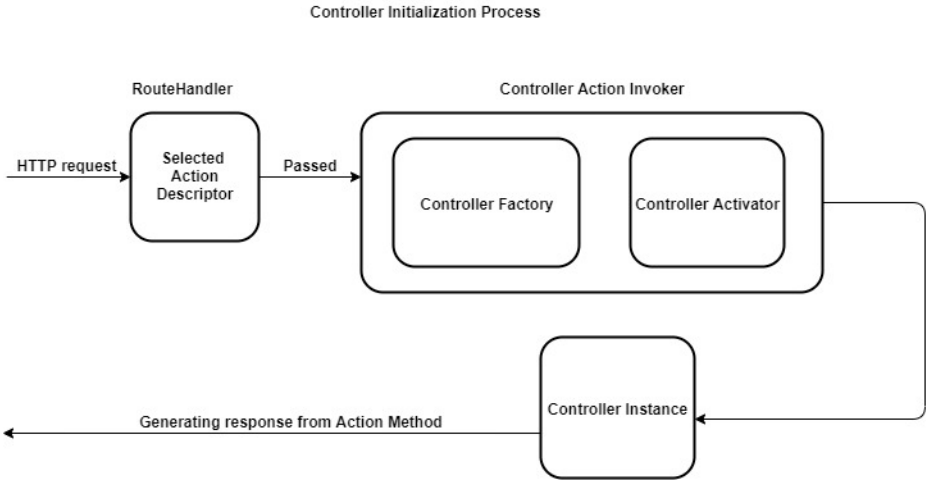
       The Exception filters are used to handle exception that occurred before anything written to the response body.
    5. Result filters :

       The Result filters are used to run code before or after the execution of controller action results. They are executed only if the controller action method has been executed successfully.

2.4 Controller Initialization

- Controllers are responsible for handling incoming requests which is done by mapping request to appropriate action method. The controller selects the appropriate action methods (to generate response) on the basis of route templates

provided. A controller class inherits from controller base class. A controller class suffix class name with the Controller keyword.



- The RouteHandler is responsible for selecting an action method candidate in the form of action descriptor. The RouteHandler then passes the action descriptor into a class called Controller action invoker. The class called Controller factory creates instance of a controller to be used by controller action method. The controller factory class depends on controller activator for controller instantiation.
- After action method is selected, instance of controller is created to handle request. Controller instance provides several features such as action methods, action filters and action result. The activator uses the controller type info property on action descriptor to instantiate the controller by name. once the controller is created, the rest of the action method execution pipeline can run.
- The controller factory is the component that is responsible for creating controller instance. The controller factory implements an interface called IControllerFactory. This interface contains two methods which are called CreateController and ReleaseController.

2.5 Action Method

- In ASP.NET Core, Action Results are the types that you return from action methods in your controller to specify how the HTTP response should be generated and sent back to the client. Action Results represent the outcome of an action and dictate what content should be included in the response. ASP.NET Core provides a variety of built-in Action Result types to handle different types of responses.
- Here are some common Action Results :

1. ViewResult :

   Represents an HTML view template that should be rendered and returned as a response.

   Typically used for rendering HTML pages to be displayed in a web browser.
2. JsonResult :

   Represents JSON data that should be serialized and returned as a response.

   Useful for serving data to client-side JavaScript applications.
3. RedirectResult :

   Redirects the client to a different URL or action method.

   Used for performing HTTP redirects.
4. ContentResult :

   Represents plain text, HTML, or any content that doesn't require view rendering.

   Allows us to specify the content and content type directly.
5. FileResult :

   Represents a file download response.

   Allows us to send files to the client, such as images, PDFs, or documents.
6. StatusCodeResult :

   Represents an HTTP status code without a specific content response.

   Used for returning HTTP status codes like 404 (Not Found), 500 (Internal Server Error), etc.