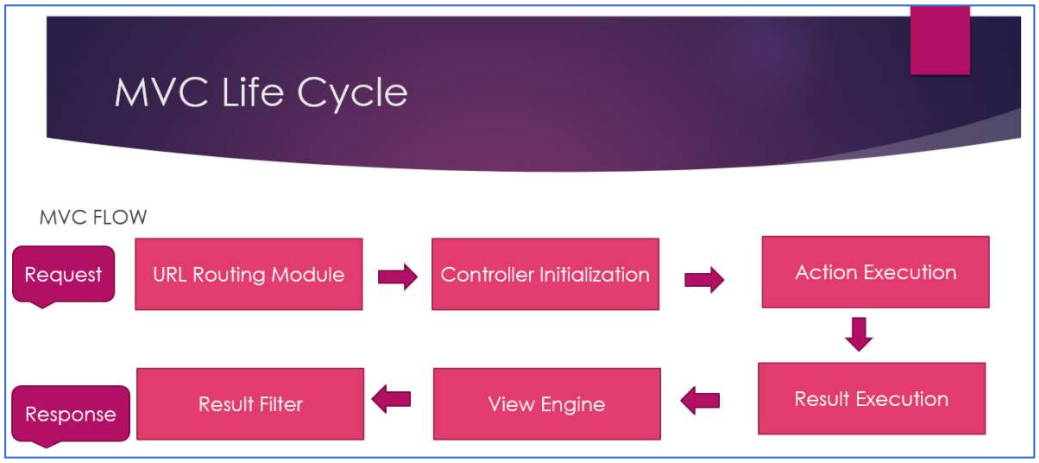
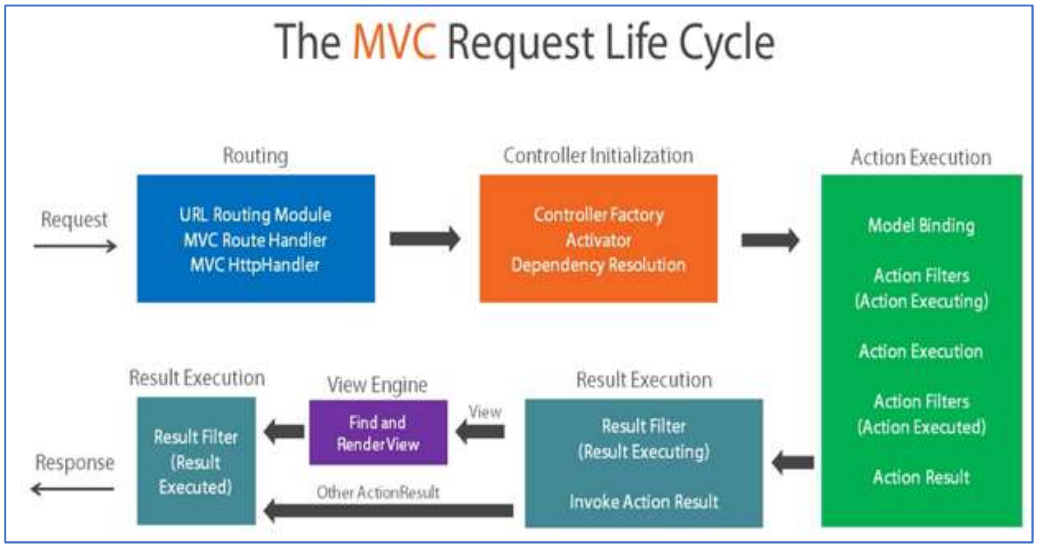
MVC Page Lifecycle

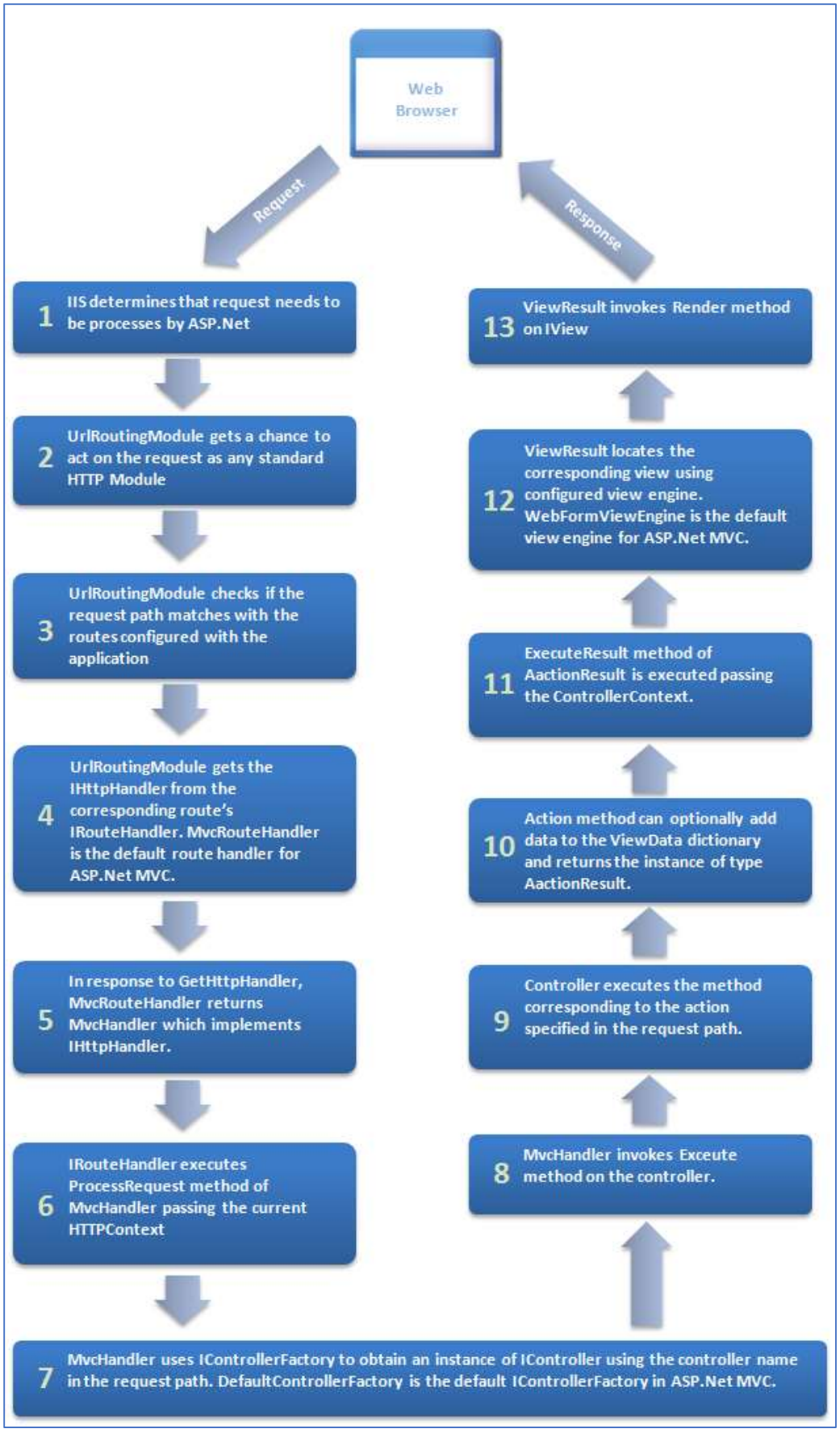
Overview





In-Depth

https://blogs.msdn.microsoft.com/varunm/2013/10/03/understanding-of-mvc-page-life-cycle/

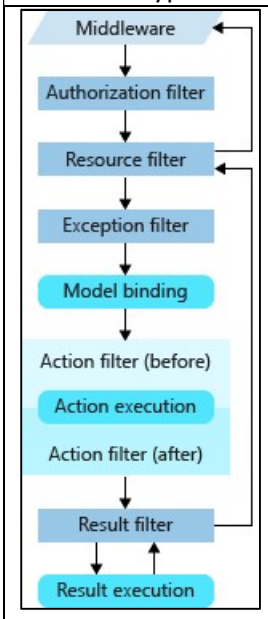
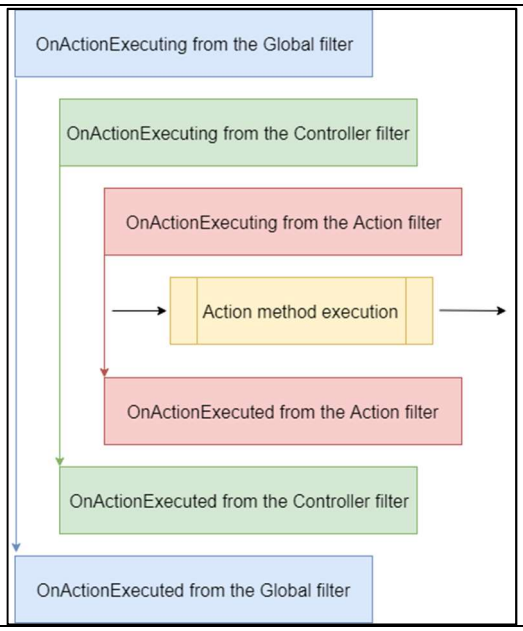

Web Browser
Request
Response
1 IIS determines that request needs to be processes by ASP.Net
2 UrlRoutingModule gets a chance to act on the request as any standard HTTP Module
3 UrlRoutingModule checks if the request path matches with the routes configured with the application
4 UrlRoutingModule gets the IHttpHandler from the corresponding route's IRouteHandler. MvcRouteHandler is the default route handler for ASP.Net MVC.
5 In response to GetHttpHandler, MvcRouteHandler returns MvcHandler which implements IHttpHandler.
6 IRouteHandler executes ProcessRequest method of MvcHandler passing the current HTTPContext
7 MvcHandler uses IControllerFactory to obtain an instance of IController using the controller name in the request path. DefaultControllerFactory is the default IControllerFactory in ASP.Net MVC.
8 MvcHandler invokes Exceute method on the controller.
9 Controller executes the method corresponding to the action specified in the request path.
10 Action method can optionally add data to the ViewData dictionary and returns the instance of type AactionResult.
11 ExecuteResult method of AactionResult is executed passing the ControllerContext.
12 ViewResult locates the corresponding view using configured view engine. WebFormViewEngine is the default view engine for ASP.Net MVC.
13 ViewResult invokes Render method on IView

MVC Filters

Each filter type is executed at a different stage in the filter pipeline:

- Authorization filters
  - They run first and are used to determine whether the user is authorized for the request. Authorization filters short-circuit the pipeline if the request is unauthorized.
- Resource filters:
  - Run after authorization.
  - `OnResourceExecuting()` can run code before the rest of the filter pipeline. For example, `OnResourceExecuting()` can run code before model binding.
  - `OnResourceExecuted()` can run code after the rest of the pipeline has completed.
- Action filters
  - They can run code immediately before and after an individual action method is called. They can be used to manipulate the arguments passed into an action and the result returned from the action. Action filters are **not** supported in Razor Pages.
- Exception filters
  - They are used to apply global policies to unhandled exceptions that occur before anything has been written to the response body.
- Result filters
  - They can run code immediately before and after the execution of individual action results. They run only when the action method has executed successfully. They are useful for logic that must surround view or formatter execution.

The following diagram shows how filter types interact in the filter pipeline.

How filter types interact in the filter pipeline	Order of Invocation:
	

The Scope of Action Filters

Like the other types of filters, the action filter can be added to different scope levels: Global, Action, Controller.

If we want to use our filter globally, we need to register it inside the `AddMvc()` method in the `ConfigureServices()` method:

```
services.AddMvc(
  config =>
  {
     config.Filters.Add(new GlobalFilterExample());
  });
```

But if we want to use our filter as a service type on the Action or Controller level, we need to register it in the same `ConfigureServices()` method but as a service in the IoC container:

```
services.AddScoped<ActionFilterExample>();
services.AddScoped<ControllerFilterExample>();
```

Finally, to use a filter registered on the Action or Controller level, we need to place it on top of the Controller or Action as a ServiceType:

```
namespace AspNetCore.Controllers
{
    [ServiceFilter(typeof(ControllerFilterExample))]
    [Route("api/[controller]")]
    [ApiController]
    public class TestController : ControllerBase
    {
        [HttpGet]
        [ServiceFilter(typeof(ActionFilterExample))]
        public IEnumerable<string> Get()
        {
            return new string[] { "example", "data" };
        }

    }
}
```