ASP – Activate Server Page

Pipeline do MVC

Usuario 🡪 IIS/ASP.NET > Rota 🡪 Inicialização de controller 🡪 execução das action 🡪

Execução das Views 🡪Com view(Renderização e inicialização de view ).

Explicação:

O ASP.NET fica hospedado pelo IIS, é a única forma de rodar.

A rota possui informações para que seja entendida a estrutura da URL requisitada e de match com as rotas obtidas no MVC

Na parte das controllers o “http handler” manipula o estado da requisição http criando uma instancia da controller factory que vai criar uma instancia da sua controller requisitada. Ele descobre a controller através da URL.

Action invoker invoca o metodo da controller que voce está chamando.

Model Binders entra em ação quando tem um model envolvida

Authentication Filter Exemplo- Se usuário está logado ou pode se logar

Authorization Filter Quais ações ele pode ter diante o login dele,validações de permissões

Action Filter ->

1° Estágio Execução da action(Action Execution(parte onde ele procura alguma model, vai no banco procura camadas na sua arquitetura) e Action Executed)

2° Action Filter – filtros para saber se o usuário ta logado ou não, se tem permissão ou não, para saber se deu erro ou não.

Result Filter 2 tipo de resultado – Sem view(passa direito e entrega o resultado final)

Com View(A view depende de elementos que estão codificados como RAZOR que ajuda quando for codificar a lógica do negócio trazendo para View)

Diferença de asp.net core e MVC 5

Asp.net core traz o pilar das inovações, open source, multiplataforma(Linux,mac,Docker, nuvem) o MVC so roda em Windows pq ele é totalmente dependente da hospedagem do IIS.

Conceito OWIN(Open web interface for .net)

Model MVC:

WEBHOST(IIS) ASP.NET COMPONENTS(ATRELADO AO IIS)

Com OWIN:

IIS, CUSTOM HOST, SELF HOST

Interface OWIN ligados os dois, não gerando dependências

ASP.NET COMPONENTS

Para que usar o ASP.NET

Sites: MVC, Web pages, web forms

Service: webapi, SIgnalR(chat, aplicações com comunicação em tempo real)

Identity: Se tiver um usuário logado

O padrão MVC:

Separação de responsabilidades

ROTAS:

As rotas mais complexas ficam em cima na parte do route, pois quando a aplicação é carregada é formado um dicionário de rotas onde quando foi requisitado alguma URL será testado em cada rota para saber se cabe, o problema é que a url e a rota podem coincidir e acabar tendo um mal funcionamento de forma em que não reconheça o que voce esteja pedindo

Exemplo:

URL: sistema/home/indexteste

{controller}/{action}/{id}

sistema/{controller}/{action}/{id}

Dessa forma a “sistema” pode ser interpretado como uma controller, home como action e indexteste como um id. Colocando no mapeamento a rota sistema em cima ele vai entender que é uma rota complexa e vai trazer o resultado correto.

----- NÃO É RECOMENDADO MUITOS MAPEAMENTOS DE ROTA----

ROTAS POR ATRIBUTO

[RoutePrefix("Rota-fixa")]

public class HomeController : Controller

{}

[Route(template:"sobre-nos")]

public ActionResult About()

{

return View();

}

Única coisa é evitar colocar rota fixa em páginas HOME, primeira página que será exibida.

Passagem de parâmetros na controller

https://localhost:44304/parametros/1/textoteste

[Route("{id:int}/{texto:maxlength(50)}")]

public ActionResult Index(int id, string texto)

{

return View();

}

Dessa forma o parâmetro está tipado como INTEIRO, caso seja passado um parâmetro nessa rota qualquer coisa diferente de INTEIRO ele irá retornar um 404 onde não foi encontrado nenhuma action válida, sendo assim evita chances de erro e passa a procurar outras action que aceite.

Caso você não queira exigir que o parâmetro seja obrigatório na passagem da ROTA

https://localhost:44304/parametros/1?texto=textoteste

[Route("{id:int}")]

public ActionResult Index(int id, string texto)

{

return View();

}

MODEL

DataAnnotations:

Determinar validação de data types

Validação de conteúdo dos dados

Validação dos formulários

[Compare(otherProperty:"ConfirmacaoEmail", ErrorMessage = "As senhas informadas não conferem")]

[DisplayName("E-mail")]

[Required(ErrorMessage = "O campo {0} é requirido")] [EmailAddress(ErrorMessage ="Email em formato inválido !")]

public string Email { get; set; }

{0} é um parâmetro que pega o próprio nome de atributo e informa o tipo de erro dado.

Compare compara o que foi informado em 2 campos, sendo o outro campo nesse caso a confirmação do email.

As validações desses dados são feitos quando através do ModelBinding que tem dentro das action que são recebidas por parâmetros através dos data annotations colocados em cada campo,

if (!ModelState.IsValid) return View(aluno);

ModelState agrupa todos os erros encontrados devolvendo a view com a model preenchida com os erros encontrados na hora da validação. É dessa forma que se compara dados informados dos usuários na hora do cadastro e salvar no banco. E dessa forma montado ele retorna os dados escritos no campos sem q precisa colocar novamente na mão ou algum input automatizado.

RAZOR

HTML HELPER

@Html.ValidationSummary(excludePropertyErrors:false, message:"",htmlAttributes: new { @class = "text-danger"})

Esse tipo de validação vai expor todos os erros de validação encontrado. Deixar o “message” em branco impede de ele já expor o texto de erro logo quando entra a primeira vez na pagina.

<div class="form-group">

@Html.LabelFor(expression: model => Model.CPF, htmlAttributes: new { @class = "control-label col-md-2" })

<div class="col-md-10">

@Html.EditorFor(expression: model => Model.CPF, additionalViewData: new { htmlAttributes = new { @class = "form-control" } })

@Html.ValidationMessageFor(expression: model => model.CPF, validationMessage: "", htmlAttributes: new { @class = "text-danger" })

</div>

</div>

Esse montado dessa forma, aparece na parte de baixo do input depois q for enviado e validado no model state

@Scripts.Render("~/bundles/jquery")

@Scripts.Render("~/bundles/bootstrap")

@RenderSection("scripts", required: false)

Essa parte do \_Layout do shared indica que tudo que chamar “scripts” vai ser renderizado por sessão ou seja por pagina, tirando que, se esse script for renderizado sem a parte “@section” ele irá renderizar antes dos script principais, pois há uma ordem para que sejam renderizado evitando problemas que impedem que esses scripts funcionem, a ordem nesse caso é a do jquery, jqueryval.validate, jqueryval.validate.unobstrusive, sendo assim:

@section Scripts

{

@Scripts.Render("~/bundles/jqueryval")

}

Colocar esse tipo de script para que seja renderizado, é uma forma de montar validação via jquery. O jquery possui um tipo de validação no caminho jquery.validate.unobstrusive.js que serve para validar os dados enquanto se é digitado não incomodando o usuário, dessa forma garante maior validação sem que seja enviado para o servidor, ou seja faz a validação na própria página, então são feitas 2 tipos de validação no front e no backend.

Partial Views

São códigos HTML que possuem pequenos trechos de codigo que podem servir para diversas ou serem redenrizadas em algum local da view inteira em determinada condição. São caracterizadas pela nomenclatura \_Partial, sendo ela única de cada View tem que ser colocada dentro da pasta da view, agora se for uma partial que pode servir para várias, tem q ser colocada dentro da pasta Shared onde vai ser localizada por todas as views. Para serem renderizada em alguma parte do codigo HTML basta adicionar o codigo razor com html helper onde voce quer que apareça o codigo Html, Exemplo caso seja um input de texto, colocar para ser redenrizado no local do formulário:

@Html.Partial(“\_NomePartialView”)

Trabalhando com rota o Html helper possui um metodo para indicar a controller e a rota que queira passar:

<a href=”@Url.Action(action:“Index”, controllername:“Home”)”></a>

Proteção contra-ataques CSRF

Cross-Site Request Forgery – Falsificaçao de solicitação entre sites.

Referência:

<https://www.eduardopires.net.br/2018/02/prevenindo-ataques-csrf-no-asp-net-core/>

É um ataque que pode ser capturadas informações sobre login de cadastros dos sites a partir de formulários enviados de forma maliciosa, onde pode ser um botão enviado por e-mail, ou algum patrocínio das páginas e etc. Esse formulário irá fazer uma requisição para o seu servidor na sua action e fazer o retorno normal da sua aplicação.

Dentro de todo formulário é importante introduzir o “@Html.AntiForgeryToken()” é um método que gera 2 códigos Hash nunca igual ao outro um vai no Body e o outro no header em forma de cookie e quando o usuário submeter esse formulário as informações que vão chegar a controller responsável terá um data annotation validador desse token “[ValidateAntyForgeryToken]” que vai conferir se esse token está correto com o que é esperado e vai processar a controller normalmente, caso contrario será lançado um erro de validação de token e o processo será encerrado.

CDN – Content Delivery Network

São lugares onde são hospedados os dados para o fornecimento para o usuário

Exemplo: Fazer o download de algum arquivo que seja da Microsoft então ao invés de eu baixar esse arquivo lá do EUA, eu baixo do lugar mais próximo que estiver hospedado

Todo browser possui um limitador de downloads simultâneos por host

Bundling – É o processo de transformar todo tipo de arquivo do mesmo tipo em apenas um so, sendo assim poupa ciclos e abre mais oportunidades para downloads do browser.

meusite.js 🡪

jquery.js 🡪 TUDO.js

funções.js 🡪

Mas também nem tudo são rosas, por as vezes tem arquivo que depende de outros, as vezes funções com o mesmo nome, nem tudo da para misturar mas dá pra misturar bastante coisa.

Minification:

É o processo de comprimir o nome dos arquivos através da remoção de espaços em branco, comentários, quebras de linha e substituição de variáveis. Ele deixa inclusive o codigo inteiro em uma linha só.

Deixar os arquivos Javascript por ultimo para renderizar

Esses bundles estão localizados no APPstart 🡪 bundleconfig.cs:

Para criar um bastar criar um metodo RegisterBundles(BundleCollection bundles)

E adicionar new ScriptBundle que recebe o caminho principal que é o arquivo que vão ser agrupados tudo que voce por no include e dps voce da um .Include() que recebe um array do caminho do dos arquivos que voce quer agrupar, no caso do jquery ele poe um {version} para que inclua todas as versões disponíveis na pasta, o jquery.validate\* tem esse \* para que seja tudo dps de validate agrupado, evitando que seja declarado um por um. O StyleBundle é a mesma coisa e o mesmo passo para adicionar, porém é um script para estilos, como boostrap e CSS, não pode misturar js com estilos

Depois de adicionar na página de \_Layout, onde renderiza para todas as páginas é adiciona o caminho principal

@Scripts.Render("~/bundles/testes")

---- IMPORTANTE ----

Em aplicações que rodam locais é necessário ativar no próprio bundleconfig.cs esse codigo:

BundleTable.EnableOptimizations = true;

Para que não precise ficar comentando codigo quando for subir em produção, da para alterar no web config a opção de debug, onde voce coloca como false e funciona como se fosse uma aplicação em produção.

Quando inspecionado, os bundles são criados jquery?v=version esse “version” é criado toda vez que recompila para quando for criado o cache ele identificar qual versão está usando para olhar as alterações.

ENTITY FRAMEWORK

MIGRATIONS:

Automatizam processos de atualização de BD

Geram scripts de BD

Realizam tarefas no BD

Versionam Atualizações no BD

Baseado em linha de comando

Facil aprendizado

PM > Enable-Migrations: Vai verificar se a connection string aponta para alguma base existente e se não ele cria uma base nova, criando uma pasta migrations com um classe:

Definir o AutomaticMigrationsEnabled na classe Configurations de migration como true para que os comandos tenham efeito.

PM> Update-Database : -Force = Força o comando com risco de perda de dados do tipo AH quero excluir essa tabela, mas ela contem dados

-Verbose = Ter uma visão mais completa do que está acontecendo ou seja mostra todo codigo que é feito por baixo dos panos;

-Script = É para gerar um script de migração de banco para atualizar o banco

PM> Add-Migration: Gera um script para criar uma migração, serve mais para ter um histórico do que foi acontecendo com o banco até o estado agora.

PM> Get-Migrations: Obtem todas as migrações do que aconteceu.

CRUD

Atualizar:

aluno.Nome = "Eduardo";

var entry = context.Entry(aluno);

context.Set<Aluno>().Attach(aluno);

entry.State = EntityState.Modified;

context.SaveChanges();

Adicionar:

context.alunos.Add(aluno);

context.SaveChanges();

Remover:

context.alunos.Remove(aluno);

context.SaveChanges();

Listar:

var alunos = context.alunos.ToList();

return View("NovoAluno", alunos.FirstOrDefault());

Configuração de ModelBuilder:

modelBuilder.Conventions.Remove<PluralizingTableNameConvention>();

modelBuilder.Entity<Aluno>().ToTable("Alunos");

Templates criadas a partir da validação de usuários quando criado o projeto

Identity config possui validações de usuário como se o login

Enviar email de cadastro, Enviar sms de cadastro verificação de codigo,

Permissão de alfanumérico, não permitir usuários com 2 emails igual, validação de senha, bloqueio de senha por tentativa de.

StartupAuth, classe partial da startup onde possui configurações de validação de usuário.

Scaffolding (andaime)

Trazer benefícios de automaç~çao

Oidem ser customizadas através de um template t4

É para habilitar e tirar atividades que são feitas manualmente

Criando esse procedimento, automaticamente é criado todo o CRUD de controller e de páginas view do modelo que você referência e o dbcontext é inserido dentro da classe IdentityModels, voce so precisa colocar os dbset e onmodelcreating, que são propriedades de customização

Processo de bind: Processo de segurança onde voce passa o nome dos atributos da classe que voce quer receber, ou seja, os campos que não estiverem setados ali não serão atribuído valor. Caso alguém manipule o seu html voce acaba não salvando dados externos indesejados, porém tudo que voce quer receber precisar estar cscrito ali.

public async Task<ActionResult> Create([Bind(Include = "Id,Nome,Email,CPF,DataMatricula,Ativo")] Alunos alunos)

Manutenção de estado:

ViewBag ViewData = Maneira de passar informações da controller para view sem ter que popular alguma Model.

Curto tempo de duração, ou seja, somente durante ou request ou até o refresh da página, após lidas perdem o conteúdo.

TempData = Controller pass para outra controller e dai chama a view, o tempdata serve para isso, o tempo de duração da informação é maior.

Depois da leitura é perdido o conteúdo.

ActionFilters

Outputcache = mantem as informações daquela controller salvas em cache pelo tempo determinado por voce, então se ocorrer updates inserts deletes no banco a alteração não vai ser vista até aquele tempo terminar, mais utilizado por conteúdos estáticos, como textos, htmls puros