ETEC PROF. HORÁCIO AUGUSTO DA SILVEIRA

DESENVOLVIMENTO DE SISTEMAS – 2022/1

Luiz Fernando Souza / Quitéria Danno

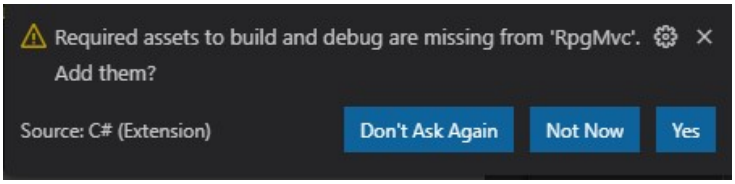
Aula 15 - Registro e login de Usuário no projeto MVC

Na última aula fizemos a publicação da API no servidor Somee e agora será o momento de criar uma aplicação com Front-end para consumir esta API e exibir os dados na página. Para isso iremos utilizar uma classe ViewModel (espécie de Model para Front-end) para que os dados da API preencham ela, uma controller para intermediar a requisição com a API e enviar os dados para a interface e páginas (Views) com extensão "*.cshtml*", que mescla códigos html e razor. Cada conjunto de operações na controller terá uma View correspondente.

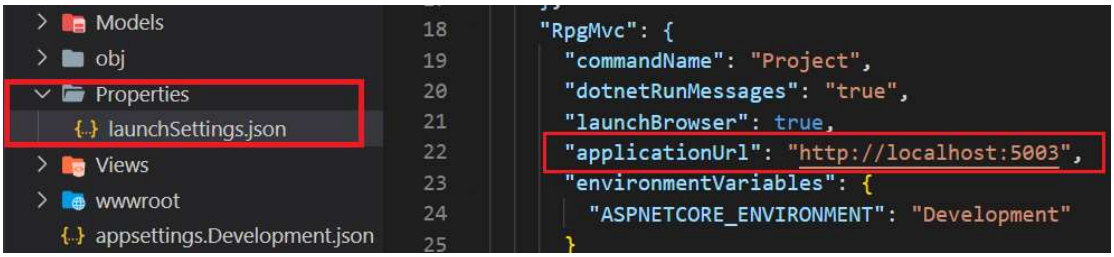
1. Navegue no Windows até a pasta onde costuma salvar seus projetos, crie uma pasta chamada **RpgMvc** e abra a mesma no VS Code.

2. Digite o comando *dotnet new MVC --framework net5.0*

3. Clique em qualquer arquivo .cs (Startup.cs) e caso a mensagem abaixo para ativar a depuração apareça, clique em Yes.



4. Abra o arquivo lauchSettings.json, dentro da pasta Properties e remova o endereço de execução *https*, mantendo apenas o endereço *http* conforme abaixo.



5. Abra a classe Startup para fazer a configuração para criação de Sessão no Método ConfigureServices

```
services.AddSession(
                option => {option.IdleTimeout = System.TimeSpan.FromSeconds(3600);}
            );
```

- Utilizaremos Session para poder armazenar o token do usuário. Session é uma variável que fica armazenada no servidor em que aplicação se encontrar e que dura um tempo determinado, expirando logo após o período.
- Note que a configuração aqui realizada é de 60 minutos (3600 segundos)

6. Adicione o uso de Sessão no método Configure da mesma classe. O trecho em destaque tem que ficar exatamente antes da configuração do UseEndPoints

```
app.UseSession();

app.UseEndpoints(endpoints =>
{
    endpoints.MapControllerRoute(
        name: "default",
        pattern: "{controller=Home}/{action=Index}/{id?}");
    endpoints.MapRazorPages();
});
```

7. Crie a classe **UsuarioViewModel** na pasta Models.

```
public class UsuarioViewModel
{
    0 references
    public int Id { get; set; }
    0 references
    public string Username { get; set; }
    0 references
    public string PasswordString { get; set; }
    0 references
    public byte[] Foto { get; set; }
}
```

8. Crie uma controller chamada **UsuariosController** dentro da pasta Controllers. As *controllers* em projetos MVC herdam de *Controller*, sendo necessário o using sinalizado acima. Acione através do atalho Control + . (ponto)

```
using Microsoft.AspNetCore.Mvc;

namespace RpgMvc.Controllers
{
    0 references
    public class UsuariosController : Controller
    {
        0 references
        public string uriBase = "xyz/Usuarios/";
        //xyz tem que ser substituído pelo endereço da sua API.
    }
}
```

9. Adicionar o pacote de conversão de Json, utilize no terminal o comando:
   dotnet add package Microsoft.AspNetCore.Mvc.NewtonsoftJson -v 5.0.15

10. Ainda na *controller*, insira o método HttpGet que irá carregar a *view* inicialmente

```
[HttpGet]
0 references
public ActionResult Index()
{
    return View("CadastrarUsuario");
}
```

11. Crie o corpo do método HttpPost. Usaremos o try/catch ao criar um método para que em caso de erro redirecionaremos para a própria index para exibir uma mensagem. exigira os usings: RpgApi.Models; System.Threading.Tasks.

```
[HttpPost]
0 references
public async Task<ActionResult> RegistrarAsync(UsuarioViewModel u)
{
    try
    {
       //Próximo código aqui
    }
    catch (System.Exception ex)
    {
        TempData["MensagemErro"] = ex.Message;
        return RedirectToAction("Index");
    }
}
```

12. Agora criaremos o método HttpPost que postará os dados para *API* para registrar o usuário. Usings HttpClient → *System.Net.Http,* JsonConvert → *Newtonsoft.Json*

```
try
{
A   HttpClient httpClient = new HttpClient();
    string uriComplementar = "Registrar";

    var content = new StringContent(JsonConvert.SerializeObject(u));
B   content.Headers.ContentType = new System.Net.Http.Headers.MediaTypeHeaderValue("application/json");
    HttpResponseMessage response = await httpClient.PostAsync(uriBase + uriComplementar, content);

C   string serialized = await response.Content.ReadAsStringAsync();

D   if (response.StatusCode == System.Net.HttpStatusCode.OK)
    {
        TempData["Mensagem"] =
            string.Format("Usuário {0} Registrado com sucesso! Faça o login para acessar.", u.Username);
        return View("AutenticarUsuario");
    }
E   else
    {
        throw new System.Exception(serialized);
    }
}
```

(A) Instância do objeto HttpClient e criação de string para trecho final da rota do método da API
(B) Serialização do objeto u. Definição do conteúdo http como Json e envio dos dados para API, guardando o retorno na variável response que armazena todos os dados do retorno da requisição.
(C) Buscando e armazenando dados de de retorno dentro do retorno da requisição
(D) Consultando qual foi o status da requisição, se foi Ok irá guardar uma mensagem temporária e redirecionar para outra a view de login.
(E) Se não for Ok lançará uma exceção com o que retornou de “serialized” para o catch

13. Crie uma pasta chamada **Usuarios** dentro da pasta Views e dentro da pasta Usuarios crie o arquivo **CadastrarUsuario.cshtml**, inserindo as tags de design abaixo

```
<!--Namespace da classe de Modelo-->
@model RpgMvc.Models.UsuarioViewModel
<!--Título da View-->
@{
    ViewBag.Title = "Registrar";
}
<!--Configuração de mensagem temporária-->
@if (@TempData["MensagemErro"] != null)
{
    <div class="alert alert-danger" role="alert">
        @TempData["MensagemErro"]
    </div>
}
<h2>Criação de novos Usuários</h2>
<hr />
<!-- Método que será chamado se o usuário fizer o Post -->
@using (Html.BeginForm("Registrar", "Usuarios", FormMethod.Post))
{
    @Html.AntiForgeryToken()
    <div class="form-horizontal">
        <div class="form-group">
            <label class="control-label col-md-2">Usuário</label>
            <div class="col-md-6">
                <!--Campo TextBox -->
                @Html.EditorFor(model => model.Username, new { htmlAttributes = new { @class = "form-
control" } })
            </div>
        </div>
        <div class="form-group">
            <label class="control-label col-md-2">Senha</label>
            <div class="col-md-6">
                <!--TextBox de Senha-->
                @Html.PasswordFor(model => model.PasswordString, new { @class = "form-control" })
            </div>
        </div>
        <input type="submit" value="Registrar" class="btn btn-primary" />
        <!--Exemplo de Link (Texto, View, Controller) -->
        @Html.ActionLink("Retornar", "Index", "Home", null, new { @class = "btn btn-warning" })
    </div>
}
```

14. Crie um arquivo chamado **_LoginPartialRpg.cshtml** dentro da pasta Views/Shared e inclua a tags abaixo

```
<ul class="navbar-nav">
    <li class="nav-item">
        <a class="nav-link text-dark" asp-area="" asp-controller="Usuarios" asp-
action="Index" >Registrar-se</a>
    </li>
    <li class="nav-item">
        <a class="nav-link text-dark" asp-area="" asp-controller="Usuarios" asp-
action="IndexLogin">Login</a>
    </li>
</ul>
```

- A pasta Shared contém *views* que podem ser utilizadas em todo projeto, e que por padrão a nomenclatura do arquivo se inicia com _ (underline). Perceba que os links estão redirecionando para a *controller* recém-criada.

15. No arquivo _Layout.cshtml, insira a tag partial, com a propriedade name depois do fechamento da ***ul***.

```
<div class="navbar-collapse collapse d-sm-inline-flex justify-content-between">
    <ul class="navbar-nav flex-grow-1">
        <li class="nav-item">
            <a class="nav-link text-dark" asp-area="" asp-controller="Home" asp-action="Index">Home</a>
        </li>
        <li class="nav-item">
            <a class="nav-link text-dark" asp-area="" asp-controller="Home" asp-action="Privacy">Privacy</a>
        </li>
    </ul>
    <partial name="_LoginPartialRpg" />
</div>
```

- Execute o projeto e perceba as diferenças. O efeito desta mudança é que na tela inicial, ao clicar em registrar-se, será exibida a área de novos usuários criada na etapa 4. Tente fazer as operações para registrar um login já existente no banco de dados e registrar um login inédito, conferindo se os dados de usuário cadastrado estão indo para o banco de dados.

- Você deve ter percebido que quando o usuário foi registrado, a página ficou em branco, isso se deve ao fato de não ter a *view* de login ainda, que é para onde o usuário será direcionado após o cadastro. Está será a próxima etapa.

16. Volte a Controller UsuariosController e adicione o método para abrir a *view* de login.

```
[HttpGet]
0 references
public ActionResult IndexLogin()
{
    return View("AutenticarUsuario");
}
```

17. Crie o corpo do método de autenticação

```
[HttpPost]
0 references
public async Task<ActionResult> AutenticarAsync(UsuarioViewModel u)
{
    try
    {
        //Próximo código aqui
    }
    catch (System.Exception ex)
    {
        TempData["MensagemErro"] = ex.Message;
        return IndexLogin();
    }
}
```

18. abaixo de para enviar os dados de login via post para a *API*. Em **SetString (A)**, será necessário o using Microsoft.AspNetCore.Http. Em **(B)** Utilize o using System.Net.Http.Headers

```
try
{
    HttpClient httpClient = new HttpClient();
    string uriComplementar = "Autenticar";

    var content = new StringContent(JsonConvert.SerializeObject(u));
    content.Headers.ContentType = new MediaTypeHeaderValue("application/json"); B
    HttpResponseMessage response = await httpClient.PostAsync(uriBase + uriComplementar, content);

    string serialized = await response.Content.ReadAsStringAsync();

    if (response.StatusCode == System.Net.HttpStatusCode.OK)
    {
      A HttpContext.Session.SetString("SessionTokenUsuario", serialized);
        TempData["Mensagem"] = string.Format("Bem-vindo {0}!!!", u.Username);
        return RedirectToAction("Index", "Personagens");
    }
    else
    {
        throw new System.Exception(serialized);
    }
}
```

- Perceba em (A) que estamos guardando o retorno da requisição numa sessão. Como testado anteriormente no *postman*, o retorno será da chamada é um Token.

19. Crie o arquivo **AutenticarUsuario.cshtml** dentro da pasta Views/Usuarios, inserindo as tags a seguir. Após isso execute o projeto para testar a página de acesso.

```
@model RpgMvc.Models.UsuarioViewModel
@{
    ViewBag.Title = "Autenticar"; }

@if (@TempData["Mensagem"] != null)
{
    <div class="alert alert-success" role="alert">
        @TempData["Mensagem"]</div>
}
<!--Configuração para exibir mensagem de erro -->
@if (@TempData["MensagemErro"] != null)
{
    <div class="alert alert-danger" role="alert">
        @TempData["MensagemErro"]</div>
}
<h2>Área de login do Usuário</h2><hr />
@using (Html.BeginForm("Autenticar", "Usuarios", FormMethod.Post))
{
    @Html.AntiForgeryToken()
    <div class="form-horizontal">
        <div class="form-group">
            <label class="control-label col-md-2">Usuário</label>
            <div class="col-md-6">
                @Html.EditorFor(model => model.Username, new { htmlAttributes = new { @class = "form-control" } })
            </div>
        </div>
        <div class="form-group">
            <label class="control-label col-md-2">Senha</label>
            <div class="col-md-6">
                @Html.PasswordFor(model => model.PasswordString, new { @class = "form-control" })
            </div>
        </div>
        <input type="submit" value="Acessar" class="btn btn-success" /><br />
        <div class="form-group">
            <!-- Outra forma de inclusir um link para uma View-->
        <p> <a href="/Usuarios">Nunca acessou o sistema? Clique aqui para registrar-se!!!</a> </p>
        </div>
    </div>
}
```

- Desafio da semana: Procurar layout bootstrap gratuitos na internet para testar na view _Layout.cshtml