FIAP

FÁBRICA DE DESENVOLVIMENTO

#20 - Segurança

FIAP

FÁBRICA DE DESENVOLVIMENTO – 2018

Prof. Thiago T. I. Yamamoto

#20 - SEGURANÇA

thiagoyama thiagoyama@gmail.com

#20 - SEGURANÇA

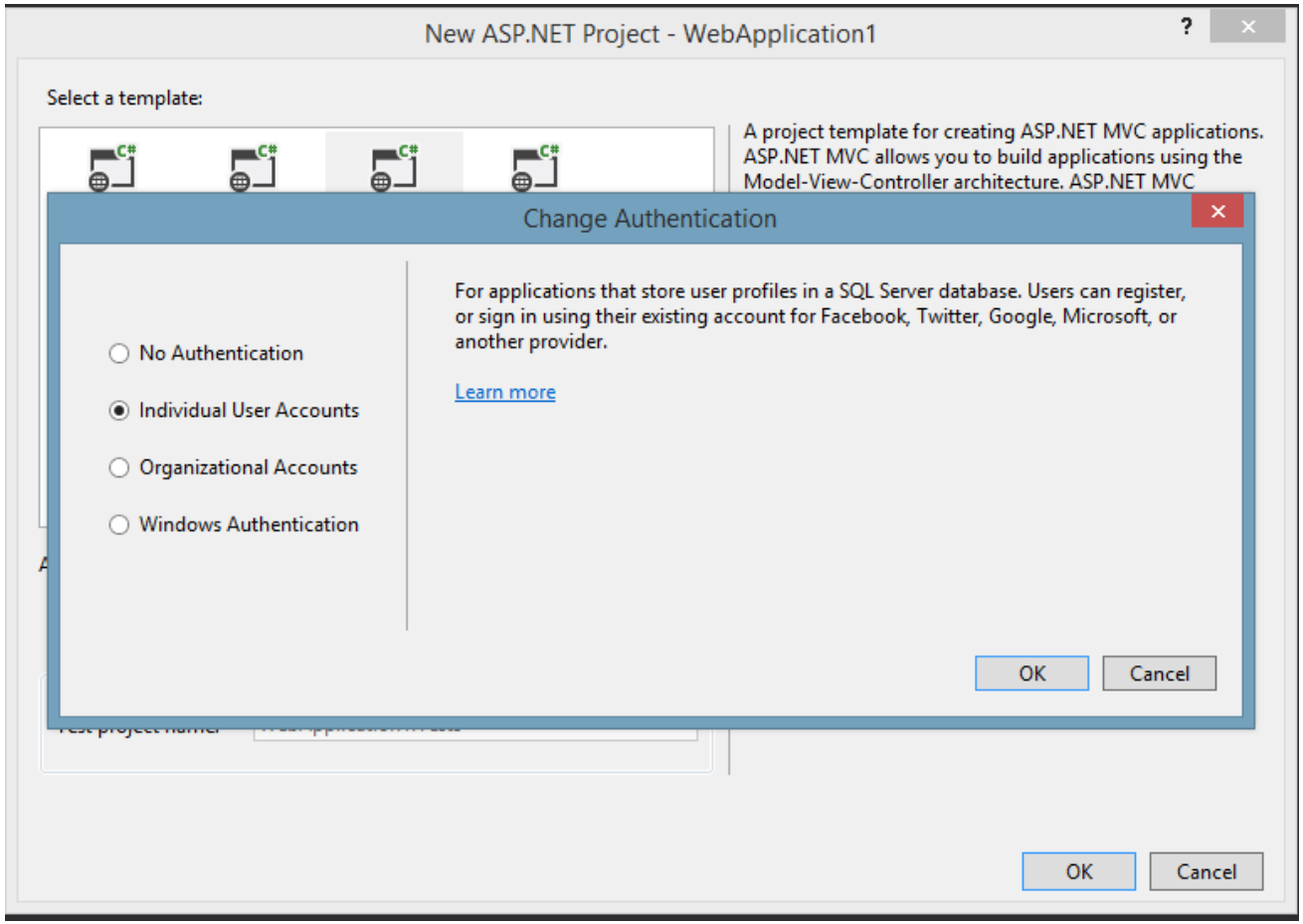
- Autenticação
- Filtros
- Configurando o projeto
  - Model e Context
  - Classe de configuração
  - Cadastrar um usuário
  - Logar
  - Logout

FIAP

AUTENTICAÇÃO

AUTENTICAÇÃO

- Podemos restringir o acesso de algumas áreas do sistema;
- **ASP.NET MVC Identity;**



http://eduardopires.net.br/2014/08/asp-net-identity-tutorial-completo/

FILTROS

- Filtros são lógicas de filtragem que são executados antes ou depois que um Action Method é chamado;
- O ASP.NET possui vários filtros prontos;
- É possível criar os nossos próprios filtros;
- Vamos utilizar **Authorization Filter**, para verificar se o usuário está logado no sistema.

```
[Authorize]
public class ClienteController : Controller
{

}
```

A anotação **[Authorize]** define que o usuário deve estar logado para acessar as actions do controller.

http://eduardopires.net.br/2013/08/asp-net-mvc-action-filters-entendendo-customizando/

FILTROS

```
[AllowAnonymous]
public ActionResult Listar()
{


}
```

A anotação **[AllowAnonymoys]** define que qualquer pessoa pode acessar a action.

```
[Authorize]
public ActionResult Cadastrar()
{


}
```

A anotação **[Authorize]** pode anotar também diretamente a action.

USER

- É possível verificar se o usuário está autenticado através da propriedade **IsAuthenticated**:

```
User.Identity.IsAuthenticated
```

- E para recuperar o nome do usuário autenticado:

```
User.Identity.Name
```

REDIRECIONANDO O USUÁRIO

- Quando o usuário tenta acessar uma área do sistema restrita, sem estar autenticado, ele é redirecionado para a página de login.
- O ASP.NET envia um parâmetro chamado **returnUrl** com o destino que o usuário tentou acessar no momento do login, com essa informação, podemos redirecionar o usuário após o login:

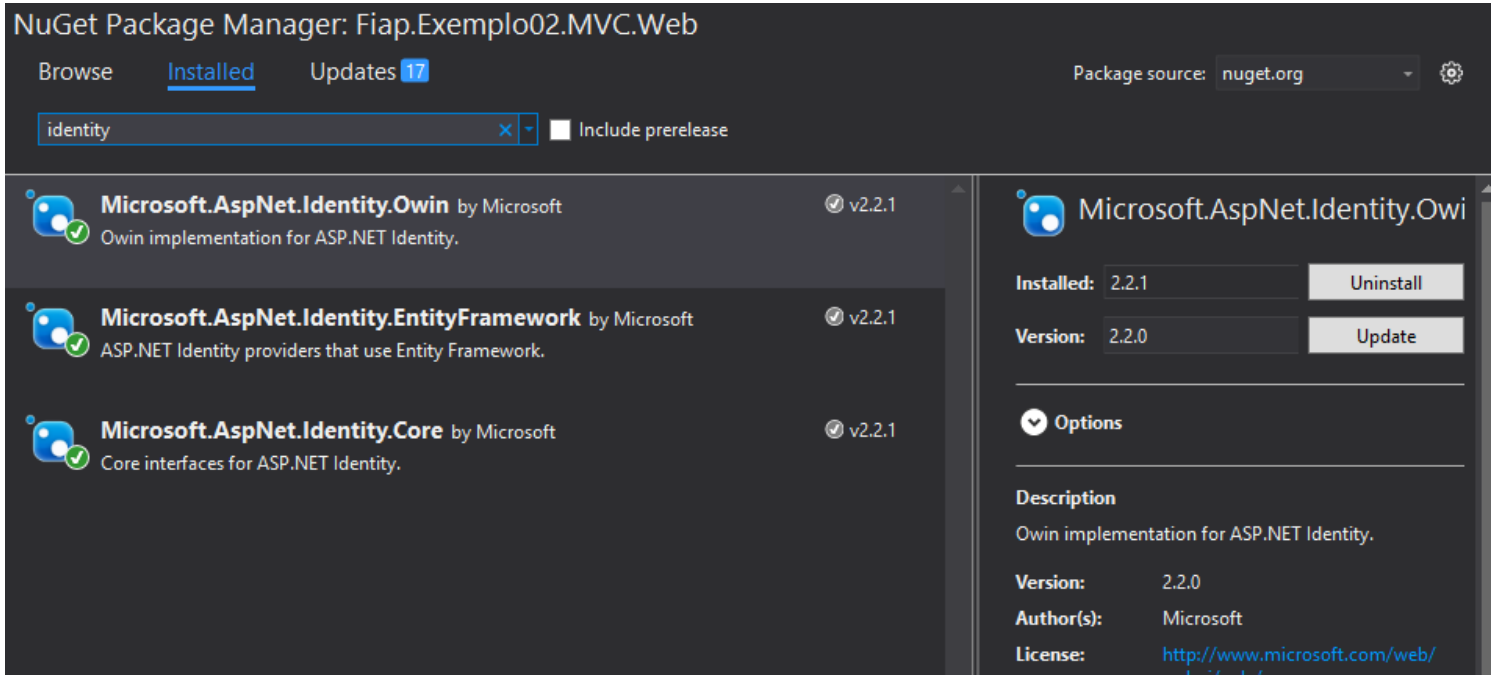
```
return Redirect(returnUrl);
```

FIAP

ADICIONANDO AUTENTICAÇÃO EM UM PROJETO EXISTENTE

CONFIGURAÇÃO

- Adicione os pactes:
  - Microsoft.AspNet.Identity.Ownin
  - Microsoft.AspNet.Identity.EntityFramework
  - Microsoft.Owin.Security.OAuth



MODEL

- Vamos criar um model de usuário que herda de **IdentyUser,** neste model e possível adicionar outras propriedades específicas.

```
public class Usuario : IdentityUser
{
    //Propriedades específicas
}
```

DBCONTEXT

- Precisamos de um contexto para gerenciar a entidade.
- O contexto herda de **IdentityDbContext** ao invés de DbContext.

```
public class UsuarioContext : IdentityDbContext<Usuario>
{
}
```

CLASSE DE CONFIGURAÇÃO

- Dentro de App_Start, vamos criar uma classe de configuração do Identity.

```
public class IdentityConfig
{
    public static Func<UserManager<Usuario>> UserManagerFactory { get; private set; }

    public void Configuration(IAppBuilder app)
    {
        app.UseCookieAuthentication(new CookieAuthenticationOptions
        {
            AuthenticationType = DefaultAuthenticationTypes.ApplicationCookie,
            LoginPath = new PathString("/usuario/login")
        });

        UserManagerFactory = () =>
        {
            var usermanager = new UserManager<Usuario>(
                new UserStore<Usuario>(new UsuarioContext()));
            // permite caracteres alfa numéricos no username
            usermanager.UserValidator = new UserValidator<Usuario>(usermanager)
            {
                AllowOnlyAlphanumericUserNames = false
            };

            return usermanager;
        };
    }
}
```

CLASSE DE CONFIGURAÇÃO

- Dentro de web.config, vamos configurar a aplicação para utilizar a classe IdentityConfig.

```
<appSettings>
    <add key="owin:AppStartup" value="Fiap.Exemplo.MVC.Web.App_Start.IdentityConfig" />
</appSettings>
```

CONTROLLER

- Vamos criar um controler para o registro, login e logout.
- O controler precisa do **UserManager** para realizar essas ações.

```
public class UsuarioController : Controller
{
    private readonly UserManager<Usuario> userManager;

    public UsuarioController()
    {
        this.userManager = IdentityConfig.UserManagerFactory.Invoke();
    }


    protected override void Dispose(bool disposing)
    {
        if (disposing && userManager != null)
        {
            userManager.Dispose();
         }
         base.Dispose(disposing);
    }
}
```

VIEWMODEL

- Não vamos utilizar o Entity do Usuário para trabalhar com as informações da tela, para isso vamos criar um View Model.

```
public class UsuarioViewModel
{
    [Required]
    [DataType(DataType.EmailAddress)]
    public string Email { get; set; }

    [Required]
    [DataType(DataType.Password)]
    public string Password { get; set; }

    [HiddenInput]
    public string ReturnUrl { get; set; }
}
```

CONTROLLER – MÉTODOS DE SUPORTE

- Vamos criar dois métodos privados no controler:
  - Para recuperar o objeto responsável por autenticar um usuário no sistema;
  - Para redirecionar usuári para uma Url.

```
private IAuthenticationManager GetAuthenticationManager()
{
    var ctx = Request.GetOwinContext();
    return ctx.Authentication;
}

private string GetRedirectUrl(string returnUrl)
{
    if (string.IsNullOrEmpty(returnUrl) || !Url.IsLocalUrl(returnUrl))
    {
        return Url.Action("index", "home");
    }

    return returnUrl;
}
```

CONTROLLER - REGISTRO

- Para cadastrar um usuário precisamos de uma action e uma tela.

```
[HttpGet]
public ActionResult Register()
{
    return View();
}
```

CONTROLLER – REGISTRO POST

```
[HttpPost]
public async Task<ActionResult> Register(UsuarioViewModel model)
{
    if (!ModelState.IsValid)
    {
        return View();
    }

    var user = new Usuario
    {
        UserName = model.Email
    };

    var result = await userManager.CreateAsync(user, model.Password);

    if (result.Succeeded)
    {
        var identity = await userManager.CreateIdentityAsync(
           user, DefaultAuthenticationTypes.ApplicationCookie);

        GetAuthenticationManager().SignIn(identity);
        return RedirectToAction("index", "home");
    }

    foreach (var error in result.Errors)
    {
        ModelState.AddModelError("", error);
    }

    return View();
}
```

CONTROLLER - LOGIN

- Vamos criar uma action para a tela de login.

```
[HttpGet]
public ActionResult LogIn(string returnUrl)
{
    var model = new UsuarioViewModel
    {
       ReturnUrl = returnUrl
    };

    return View(model);
}
```

CONTROLLER – LOGIN POST

```
[HttpPost]
public async Task<ActionResult> LogIn(UsuarioViewModel model)
{
    if (!ModelState.IsValid)
    {
        return View();
    }

    var user = await userManager.FindAsync(model.Email, model.Password);

    if (user != null)
    {
        var identity = await userManager.CreateIdentityAsync(
            user, DefaultAuthenticationTypes.ApplicationCookie);

        GetAuthenticationManager().SignIn(identity);

        return Redirect(GetRedirectUrl(model.ReturnUrl));
    }

    ModelState.AddModelError("", "Usuário e/ou Senha inválidos");
    return View();
}
```

CONTROLLER – LOGOUT

```
public ActionResult LogOut()
{
    GetAuthenticationManager().SignOut(DefaultAuthenticationTypes.ApplicationCookie);
    return RedirectToAction("index", "home");
}
```

FIAP

"Se você realmente quer algo, não espere. Ensine você mesmo a ser impaciente"
Gurbaksh Chahal