Macoratti.net ASP .NET - Introdução ao ASP .NET Identity

No artigo de hoje vou apresentar os conceitos básicos relacionados ao novo componente de **membership ASP .NET Identity.**

O sistema **ASP .NET membership** foi introduzido com a ASP .NET 2.0 em 2005 e desde então muita coisa mudou na forma como as aplicações web tratam com autenticação e autorização.

Para você ter uma visão da evolução relacionada com essa tecnologia temos a abaixo uma cronologia das atualizações:

- **2005/2010 - Membership Provider**
- **2010/2012 - Simple Membership**
- **2012/2013 - Universal Providers**
- **2013/2015 - ASP.NET Identity**

O **ASP.NET Identity** pode ser visto como uma atualização sobre o que o sistema membership deve ser quando você está construindo aplicações modernas para a web, telefone ou tablet.

Nesse contexto o **ASP.NET Identity** foi concebido para substituir os sistemas ASP .NET Membership e Simple Membership. Ele inclui suporte a perfil, integração **OAuth**, trabalha com **Owin**, e está incluído com os modelos ASP.NET no Visual Studio 2013.

Dentre as características do ASP .NET Identity podemos destacar:

- Customização do perfil do usuário simplificado (*escrito em Code First(EF)*)
- Controle de persistência de dados (*Entity Framework, NHibernate, MySQL, MongoDB, etc*)
- Totalmente testável (*Testes unitários*)
- Role Provider (*separação de acessos por perfil*)
- Claims Based *(baseado em declarações)*
- Autenticação com redes sociais (*FaceBook, Twitter,etc.*)
- Integração com Active Directory
- Integração com OWIN (*Open Web Interface for .NET*)
- Entregue via NuGet
- Open Source - Acompanhe no GitHub - https://github.com/aspnet/Identity

O ASP.NET Identity funciona com *ASP.NET MVC, WebAPI, WebPages, WebForms e SignalR*, enfim, com todos os projetos ASP .NET. (*No VS 2013 ele é usado nos modelos de projeto para ASP .NET MVC, Web Forms, Web API e SPA*)

Este artigo pretende ser um tutorial passo a passo que ilustra como os modelos de projeto usam o ASP.NET Identity para registrar, fazer o login e o logout de um usuário.

Obs: A versão atual do ASP .NET Identity é a 2.2.0 (março/2015).

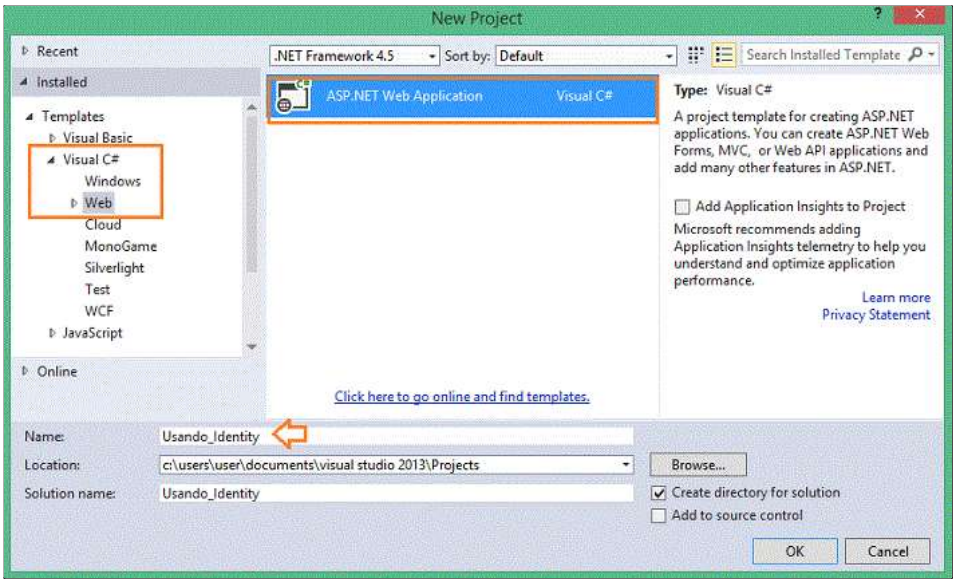
Recursos usados:

- Visual Studio Express 2013 for Web

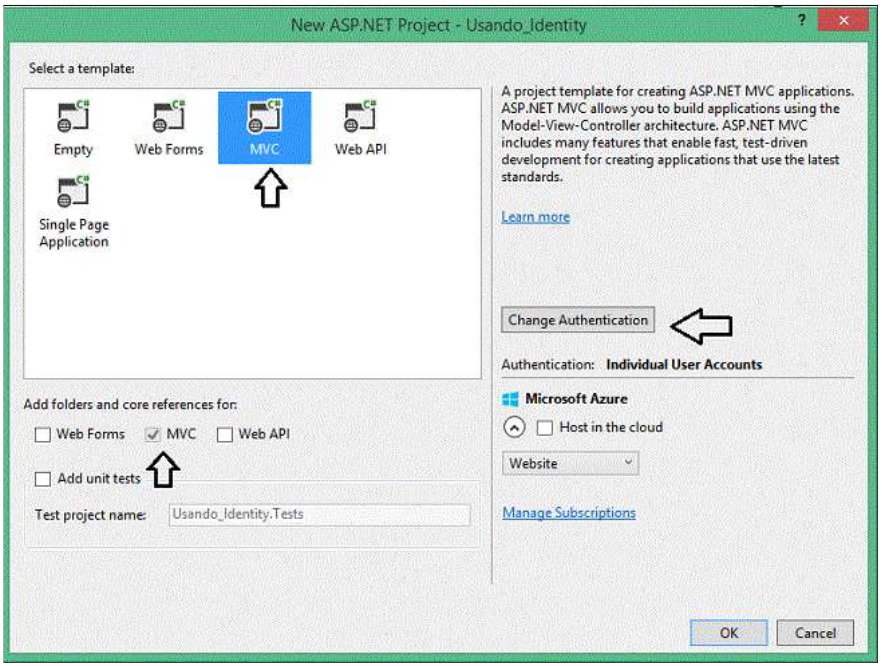
Usando o ASP .NET Identity

Então abra o VS Express 2013 for web clique em **New Project;**

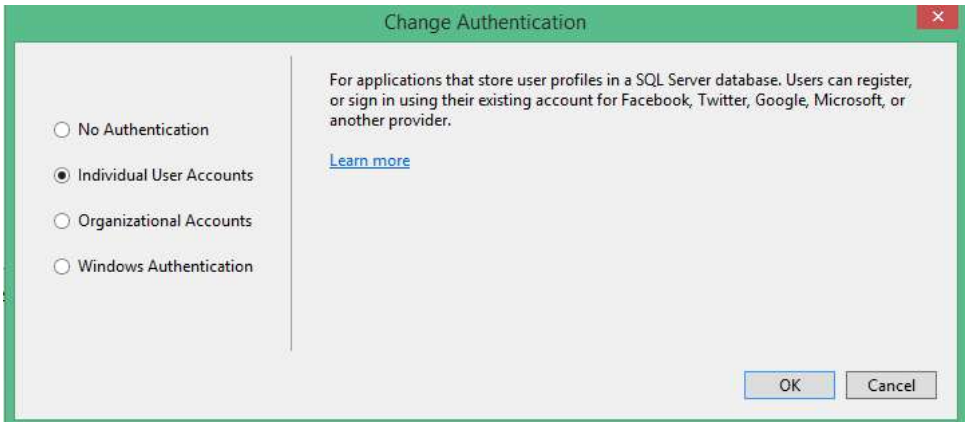
Selecione a linguagem C# e o template **Web -> ASP .NET Web Applicatoin** e informe o nome **Usando_Identity;**



A seguir selecione o template **MVC** e clique no botão **Change Authentication** para visualizar os tipos de autenticações que temos disponíveis:



Ao clicar no botão **Change Authentication** veremos a seguinte janela exibindo as autenticações possíveis e permitindo que você escolha qual usar:



Vemos os seguintes tipos de autenticação disponíveis:

- No Authentication - (*Sem Autenticação*)
- Individual User Accounts - Contas de usuários individuais (*Identity ASP.NET, anteriormente conhecido como ASP.NET membership*)
- Organizational Accounts - Contas Organizacionais (*Windows Active Directory ou Azure Active Directory Server*)
- Windows Authentication - Autenticação do Windows (*Intranet*)

O novo membership baseia-se em **Owin** em vez de o módulo de autenticação de formulários ASP.NET. Isso significa que você pode usar o mesmo mecanismo de autenticação se você está usando Web Forms ou MVC em IIS, ou em uma auto-hospedagem com API Web ou SignalR.

O novo banco de dados membership é gerenciado pelo **Entity Framework Code first**, e todas as tabelas são representados por classes de entidade que você pode modificar. Isso significa que você pode facilmente personalizar o esquema do banco de dados para atender às suas próprias necessidades, e você pode facilmente implantar suas atualizações usando **Code First Migrations**.

Vamos usar a opção marcada como **Individual User Accounts**. Clique no botão OK e a seguir confirme a opção clicando no botão OK para criar o projeto.

Ao selecionar esta opção a aplicação será configurada para usar o ASP .NET Identity ASP.NET para autenticação do usuário.

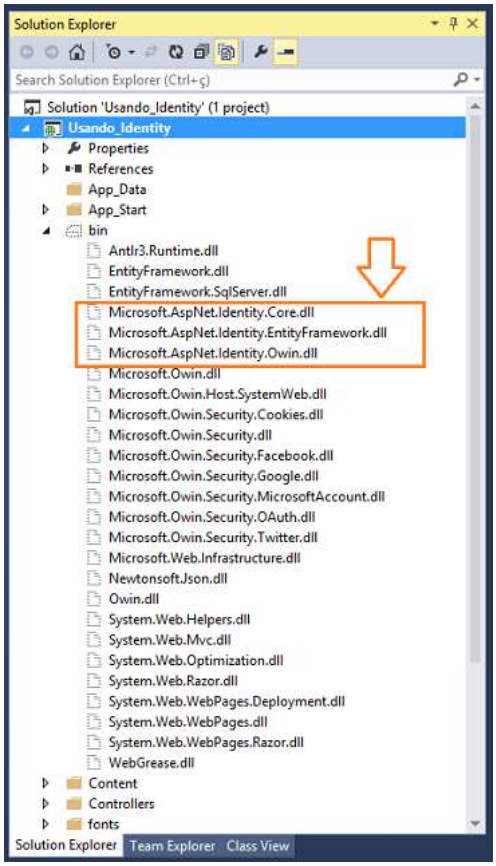
O ASP .NET Identity permite a um usuário registrar uma conta, através da criação de um nome de usuário e senha no site ou através da assinatura com provedores sociais, como Facebook, Google, Microsoft Account, ou Twitter.

O armazenamento de dados padrão para perfis de usuário no ASP .NET Identity é um banco de dados SQL Server LocalDB, que você pode implantar no SQL Server ou no SQL Azure Database para o local de produção.

Após criar o projeto vamos visualizar suas referências na janela Solution Explorer.

O projeto criado contém os três pacotes principais do **ASP .NET Identity** :

1- Microsoft.AspNet.Identity.EntityFramework



Este pacote tem a implementação Entity Framework do ASP .NET Identity que irá persistir os dados e o schema do ASP .NET Identity para o SQL Server.

2- Microsoft.AspNet.Identity.Core

Este pacote contém as interfaces fundamentais para o ASP .NET Identity e pode ser usado para escrever uma implementação para o ASP .NET Identity que tem como objetivo diferentes lojas de persistência, como Azure Storage Table, bancos de dados NoSQL etc.

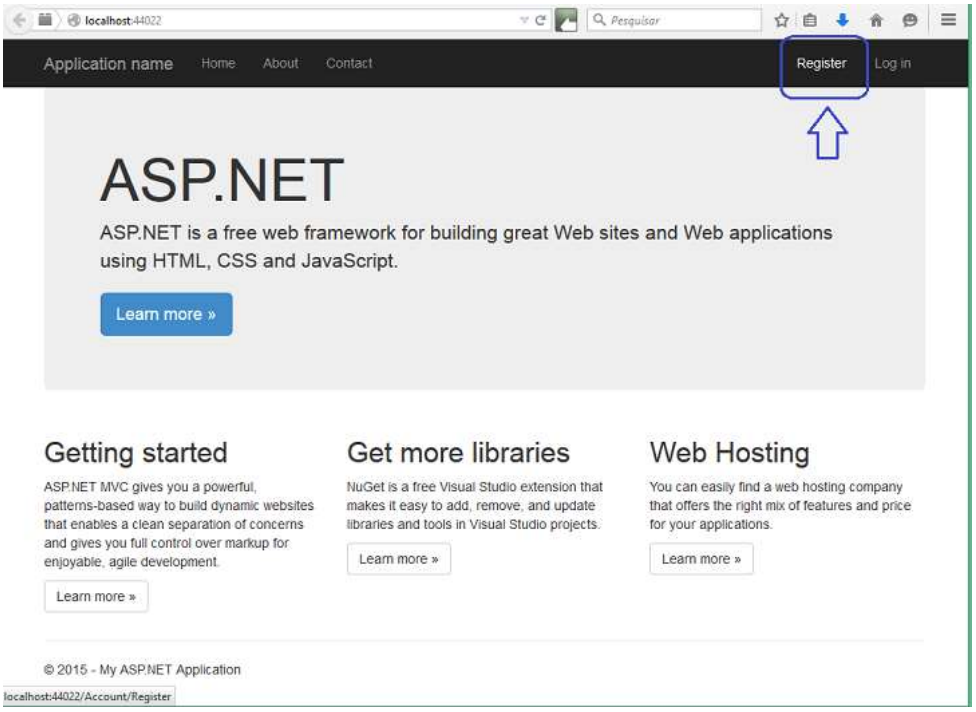
3- Microsoft.AspNet.Identity.OWIN

Este pacote contém uma funcionalidade que é usada para conectar a autenticação **Owin** com o ASP .NET Identity em aplicações ASP.NET. Isso é usado quando você adiciona a funcionalidade de log à sua aplicação e faz uma chamada ao OWIN Cookie Authentication para gerar um *cookie*.

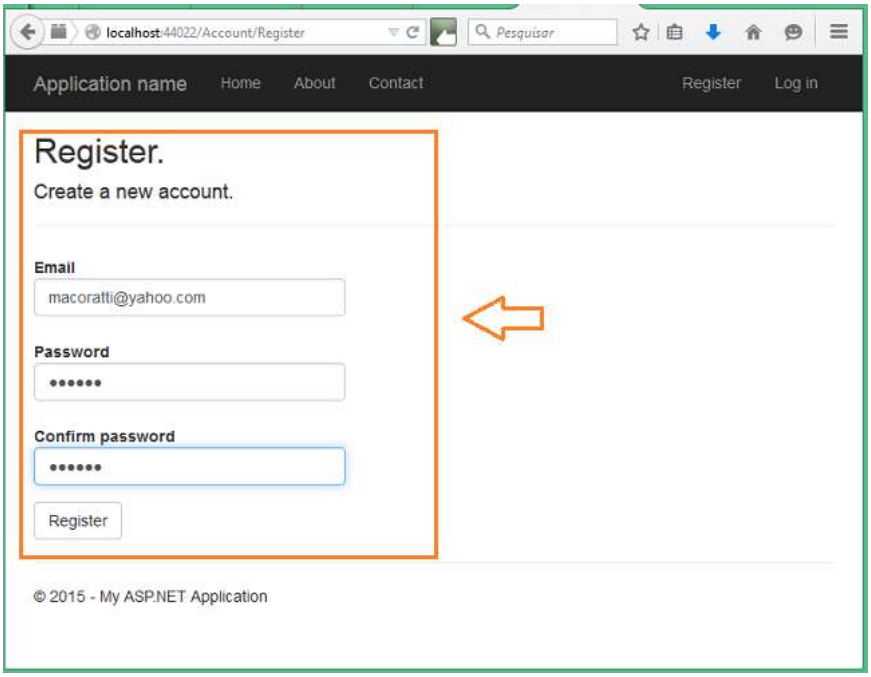
Nota: Owin - Open Web Interface for .NET - *Define uma interface padrão entre servidores web .NET e aplicações web. O objetivo da interface Owin é desacoplar o servidor e aplicação, incentivar o desenvolvimento de módulos simples para o desenvolvimento .NET web, e, por ser um padrão aberto, estimular o ecossistema open source de ferramentas de desenvolvimento web .NET.*
fonte : http://owin.org/

Testando o ASP .NET Identity

Vamos executar o projeto e criar um registrar um novo usuário clicando no botão **Register**:



Ao clicar no botão **Register** do menu será apresentada a página abaixo onde o usuário deverá informar os seus dados :



Ao clicar no botão **Register** deste formulário a Action **Register** do controlador **AccountController** que esta na pasta **\Controllers** será executada criando o usuário através da chamada da API ASP .NET Identity.

Abaixo vemos o código que realiza esta tarefa destacado em azul:

```
[HttpPost]
[AllowAnonymous]
[ValidateAntiForgeryToken]
public async Task<ActionResult> Register(RegisterViewModel model)
{
    if (ModelState.IsValid)
    {
        var user = new ApplicationUser { UserName = model.Email, Email = model.Email };
      var result = await UserManager.CreateAsync(user, model.Password);
        if (result.Succeeded)
        {
            await SignInManager.SignInAsync(user, isPersistent:false, rememberBrowser:false);

            // For more information on how to enable account confirmation and password reset please visit http://go.microsoft.co
            // Send an email with this link
            // string code = await UserManager.GenerateEmailConfirmationTokenAsync(user.Id);
            // var callbackUrl = Url.Action("ConfirmEmail", "Account", new { userId = user.Id, code = code }, protocol: Request.Url.
            // await UserManager.SendEmailAsync(user.Id, "Confirm your account", "Please confirm your account by clicking <a href

            return RedirectToAction("Index", "Home");
        }
        AddErrors(result);
    }

    // If we got this far, something failed, redisplay form
    return View(model);
}
```

Se o usuário foi criado com sucesso ele é logado pelo método **SingInAsync** conforme o código em azul a seguir:

```
[HttpPost]
[AllowAnonymous]
[ValidateAntiForgeryToken]
public async Task<ActionResult> Register(RegisterViewModel model)
{
    if (ModelState.IsValid)
    {
        var user = new ApplicationUser { UserName = model.Email, Email = model.Email };
         var result = await UserManager.CreateAsync(user, model.Password);
        if (result.Succeeded)
        {
            await SignInManager.SignInAsync(user, isPersistent:false, rememberBrowser:false);

            // For more information on how to enable account confirmation and password reset please visit http://go.microsoft.co
            // Send an email with this link
            // string code = await UserManager.GenerateEmailConfirmationTokenAsync(user.Id);
            // var callbackUrl = Url.Action("ConfirmEmail", "Account", new { userId = user.Id, code = code }, protocol: Request.Url.
            // await UserManager.SendEmailAsync(user.Id, "Confirm your account", "Please confirm your account by clicking <a href

            return RedirectToAction("Index", "Home");
        }
```

```
            AddErrors(result);
        }

        // If we got this far, something failed, redisplay form
        return View(model);
    }
```

O código no método **SignInAsync** gera um **ClaimsIdentity**. Como o **ASP .NET Identity** e a **autenticação de cookie Owin** são baseadas em declarações do sistema *(claims-based System)*, o framework requer que o aplicativo gere um **ClaimsIdentity** para o usuário.

O **ClaimsIdentity** tem informações sobre todas as reivindicações para o usuário, tais como quais os perfis a que o usuário pertence. Você também pode adicionar mais reivindicações para o usuário nesta fase.

Segurança baseada em declarações (**claims**)

A identidade baseada em declarações(**Claims**) é um conjunto de reivindicações. Um **claim** é uma declaração/reivindicação que uma entidade *(um usuário ou outro aplicativo)* faz sobre si mesma, sendo apenas uma reclamação. Por exemplo, uma lista de reivindicação pode ter o nome do usuário, usuário de e-mail, a idade do usuário, autorização do usuário para uma ação.

Em um cenário onde temos a segurança baseada em função, um usuário apresenta as credenciais diretamente para o aplicativo, já no modelo baseado em **Claims** (Declarações), o usuário apresenta as reivindicações e não as credenciais para o aplicativo.

Para uma declaração/reivindicação ter valor prático, ela deve vir de uma entidade na qual a aplicação confia.

Clicando no botão Log off chama a Action **LogOff**() do controlador **AccountController**

```
[HttpPost]
[ValidateAntiForgeryToken]
public ActionResult LogOff()
{
        AuthenticationManager.SignOut();
        return RedirectToAction("Index", "Home");
}
```

Este código mostra o método **Owin** **AuthenticationManager.SignOut();** que é similar ao método **FormsAuthentication.SignOut** usado pelo módulo **FormsAuthentication** nas aplicações Web Forms.

Esta foi uma pequena introdução ao **ASP .NET Identity** onde apresentei o básico sobre o componente.

Podemos fazer customizações com o ASP .NET **I**dentity e realizar outras tarefas que serão abordadas em outros artigos sobre o assunto.

Pegue o projeto aqui : *Usando_Identity.zip (sem as referências)*

De sorte que haja em vós o mesmo sentimento que houve também em Cristo Jesus,
Que, sendo em forma de Deus, não teve por usurpação ser igual a Deus,
Mas esvaziou-se a si mesmo, tomando a forma de servo, fazendo-se semelhante aos homens;
E, achado na forma de homem, humilhou-se a si mesmo, sendo obediente até à morte, e morte de cruz.
Filipenses 2:5-8

- **Curso - Gerando Relatórios com o ReportViewer no VS 2013 - Vídeo Aulas**

Gostou ? Compartilhe no Facebook Compartilhe no Twitter

Referências:

- Seção VB .NET do Site Macoratti.net
- Super DVD .NET - A sua porta de entrada na plataforma .NET
- Super DVD Vídeo Aulas - Vídeo Aula sobre VB .NET, ASP .NET e C#
- Seção C# do site Macoratti.net
- Super DVD C#
- Super DVD Visual Basic
- Curso Básico VB .NET - Vídeo Aulas
- Curso C# Básico - Vídeo Aulas
- ASP.NET - Usando Application Services - Macoratti.net
- ASP.NET - Usando Roles e MemberShip - Macoratti.net
- Usando MemberShip e Roles em aplicações Windows Forms
- ASP .NET - Usando MemberShip, Roles e Profiles com o ...
- WebMatrix - Segurança e MemberShip - Macoratti.net
- Owin
- Katana

José Carlos Macoratti