## Урок 6. Авторизация.

Цель урока: Изучить способ авторизации через Cookie, использование стандартных атрибутов доступа к контроллеру и методу контроллера. Использование IPrincipal. Создание собственного модуля (IHttpModule) и собственного фильтра IActionFilter.

*Небольшое отступление: На самом деле в asp.net mvc все учебники рекомендуют пользоваться уже придуманной системой авторизации, которая называется* ***AspNetMembershipProvider****, она была описана в статье* <http://habrahabr.ru/post/142711/>*, но как-то там это объяснено с точки зрения – «нажимай и не понимай что там внутри». При первом знакомстве с asp.net mvc меня это смутило. Далее в этой статье* <http://habrahabr.ru/post/143024/> *- сказано, что пользоваться этим провайдером – нельзя.*  *Тут же мы достаточно глубоко изучаем всякие хитрые asp.net mvc стандартные приемы, так что это один из основных уроков.*

### Кукисы

Кукисы – это часть информации, отсылаемая сервером браузеру, и которую браузер возвращает обратно серверу вместе с каждым (почти каждым) запросом.

Сервер в заголовок ответа пишет:

Set-Cookie: value[; expires=date][; domain=domain][; path=path][; secure]

Например:

HTTP/1.1 200 OK  
Content-type: text/html  
Set-Cookie: name=value  
Set-Cookie: name2=value2; Expires=Wed, 09-Jun-2021 10:18:14 GMT

Браузер (если не истекло время действия кукиса) при каждом запросе:

GET /spec.html HTTP/1.1  
Host: www.example.org  
Cookie: name=value; name2=value2  
Accept: \*/\*

Устанавливаем cookie:

public ActionResult Index()

{

var cookie = new HttpCookie()

{

Name ="test\_cookie",

Value = DateTime.Now.ToString("dd.MM.yyyy"),

Expires = DateTime.Now.AddMinutes(10),

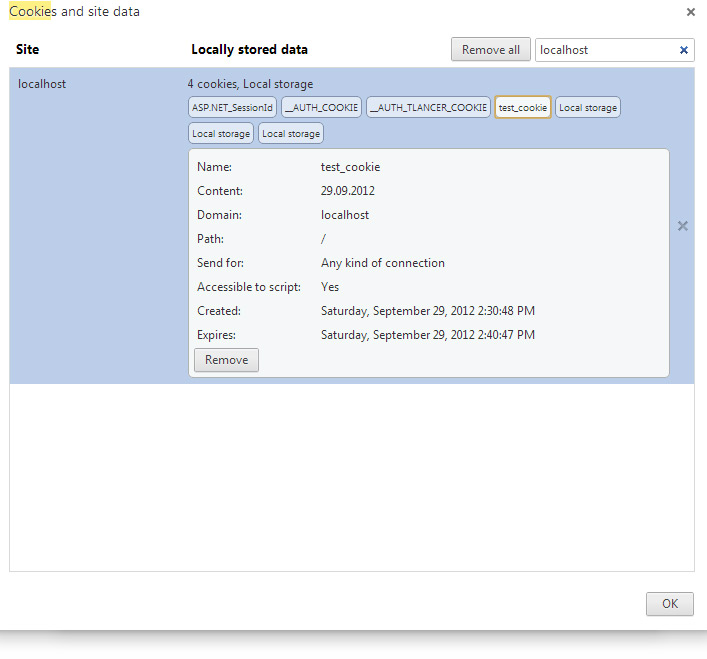
};

Response.SetCookie(cookie);

return View();

}

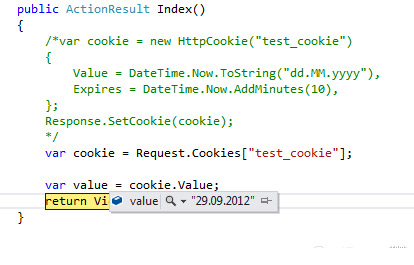
В Chrome проверяем установку:



Для получения кукисов:

var cookie = Request.Cookies["test\_cookie"];

Делаем точку останова и проверяем:



*Примечание: Тут подробнее можно изучить кукисы:*

<http://www.nczonline.net/blog/2009/05/05/http-cookies-explained/>

### Авторизация

Ок, в нашем случае авторизация будет основана на использовании кукисов. Поэтому изучим следующие положения:

* **FormsAuthenticationTicket** – мы воспользуемся этим классом, чтобы не хранить данные в открытом виде
* Нужно реализовать интерфейс **IPrincipal** и установить в HttpContext.User для проверки ролей и IIdentity интерфейса.
* Для интерфейса **IIdentity** сделать реализацию
* Вывести в **BaseController** в свойство **CurrentUser** значение пользователя, который сейчас залогинен.

Приступим.

Создадим интерфейс IAuthentication и его реализацию CustomAuthentication

public interface IAuthentication

{

/// <summary>

/// Конекст (тут мы получаем доступ к запросу и кукисам)

/// </summary>

HttpContext HttpContext { get; set; }

User Login(string login, string password, bool isPersistent);

User Login(string login);

void LogOut();

IPrincipal CurrentUser { get; }

}

Реализация:

public class CustomAuthentication : IAuthentication

{

private static NLog.Logger logger = NLog.LogManager.GetCurrentClassLogger();

private const string cookieName = "\_\_AUTH\_COOKIE";

public HttpContext HttpContext { get; set; }

[Inject]

public IRepository Repository { get; set; }

#region IAuthentication Members

public User Login(string userName, string Password, bool isPersistent)

{

User retUser = Repository.Login(userName, Password);

if (retUser != null)

{

CreateCookie(userName, isPersistent);

}

return retUser;

}

public User Login(string userName)

{

User retUser = Repository.Users.FirstOrDefault(p => string.Compare(p.Email, userName, true) == 0);

if (retUser != null)

{

CreateCookie(userName);

}

return retUser;

}

private void CreateCookie(string userName, bool isPersistent = false)

{

var ticket = new FormsAuthenticationTicket(

1,

userName,

DateTime.Now,

DateTime.Now.Add(FormsAuthentication.Timeout),

isPersistent,

string.Empty,

FormsAuthentication.FormsCookiePath);

// Encrypt the ticket.

var encTicket = FormsAuthentication.Encrypt(ticket);

// Create the cookie.

var AuthCookie = new HttpCookie(cookieName)

{

Value = encTicket,

Expires = DateTime.Now.Add(FormsAuthentication.Timeout)

};

HttpContext.Response.Cookies.Set(AuthCookie);

}

public void LogOut()

{

var httpCookie = HttpContext.Response.Cookies[cookieName];

if (httpCookie != null)

{

httpCookie.Value = string.Empty;

}

}

private IPrincipal \_currentUser;

public IPrincipal CurrentUser

{

get

{

if (\_currentUser == null)

{

try

{

HttpCookie authCookie = HttpContext.Request.Cookies.Get(cookieName);

if (authCookie != null && !string.IsNullOrEmpty(authCookie.Value))

{

var ticket = FormsAuthentication.Decrypt(authCookie.Value);

\_currentUser = new UserProvider(ticket.Name, Repository);

}

else

{

\_currentUser = new UserProvider(null, null);

}

}

catch (Exception ex)

{

logger.Error("Failed authentication: " + ex.Message);

\_currentUser = new UserProvider(null, null);

}

}

return \_currentUser;

}

}

#endregion

}

Суть сводится к следующему, мы при инициализации запроса получаем доступ к HttpContext.Request.Cookies и инициализируем UserProvider:

var ticket = FormsAuthentication.Decrypt(authCookie.Value);

\_currentUser = new UserProvider(ticket.Name, Repository);

Для авторизации в IRepository добавлен новый метод IRepository.Login. Реализация в SqlRepository:

public User Login(string email, string password)

{

return Db.Users.FirstOrDefault(p => string.Compare(p.Email, email, true) == 0 && p.Password == password);

}

UserProvider собственно реализует интерфейс IPrincipal (в котором есть проверка ролей и доступ к IIdentity).

Рассмотрим класс UserProvider:

public class UserProvider : IPrincipal

{

private UserIndentity userIdentity { get; set; }

#region IPrincipal Members

public IIdentity Identity

{

get

{

return userIdentity;

}

}

public bool IsInRole(string role)

{

if (userIdentity.User == null)

{

return false;

}

return userIdentity.User.InRoles(role);

}

#endregion

public UserProvider(string name, IRepository repository)

{

userIdentity = new UserIndentity();

userIdentity.Init(name, repository);

}

public override string ToString()

{

return userIdentity.Name;

}

Наш UserProvider знает про то, что его IIdentity классом есть UserIdentity, а поэтому знает про класс User, внутри которого мы реализуем метод InRoles(role):

public bool InRoles(string roles)

{

if (string.IsNullOrWhiteSpace(roles))

{

return false;

}

var rolesArray = roles.Split(new[] { "," }, StringSplitOptions.RemoveEmptyEntries);

foreach (var role in rolesArray)

{

var hasRole = UserRoles.Any(p => string.Compare(p.Role.Code, role, true) == 0);

if (hasRole)

{

return true;

}

}

return false;

}

В метод InRoles мы ожидаем что придет запрос о ролях, которые допущены к ресурсу, разделенные запятой, т.е. например “admin,moderator,editor”, если хотя бы одна из ролей есть у нашего User – то возвращаем – истину, доступ у него есть. Сравниваем по полю Role.Code, а не Role.Name.

Рассмотрим класс UserIdentity:

public class UserIndentity : IIdentity

{

public User User { get; set; }

public string AuthenticationType

{

get

{

return typeof(User).ToString();

}

}

public bool IsAuthenticated

{

get

{

return User != null;

}

}

public string Name

{

get

{

if (User != null)

{

return User.Email;

}

//иначе аноним

return "anonym";

}

}

public void Init(string email, IRepository repository)

{

if (!string.IsNullOrEmpty(email))

{

User = repository.GetUser(email);

}

}

}

В IRepository добавляем новый метод GetUser(email). Реализация для SqlRepository.GetUser():

public User GetUser(string email)

{

return Db.Users.FirstOrDefault(p => string.Compare(p.Email, email, true) == 0);

}

Ок, всё почти готово, выведем CurrentUser в BaseController:

[Inject]

public IAuthentication Auth { get; set; }

public User CurrentUser

{

get

{

return ((UserIndentity)Auth.CurrentUser.Identity).User;

}

}

Да, это немного не правильно, так как тут сильное связывание. Поэтому сделаем так. Введем еще один интерфейс IUserProvider, с которого мы будем требовать вернуть нам авторизованного User:

public interface IUserProvider

{

User User { get; set; }

}

…

public class UserIndentity : IIdentity, **IUserProvider**

{

/// <summary>

/// Текщий пользователь

/// </summary>

public User User { get; set; }

…

[Inject]

public IAuthentication Auth { get; set; }

public User CurrentUser

{

get

{

return ((IUserProvider)Auth.CurrentUser.Identity).User;

}

}

Ок, теперь попробуем инициализировать это всё.

Вначале добавим наш IAuthentication + CustomAuthentication в регистрацию к Ninject:

kernel.Bind<IAuthentication>().To<CustomAuthentication>().InRequestScope();

Потом создадим модуль, который будет будет на событие AuthenticateRequest совершать действие авторизации:

public class AuthHttpModule : IHttpModule

{

public void Init(HttpApplication context)

{

context.AuthenticateRequest += new EventHandler(this.Authenticate);

}

private void Authenticate(Object source, EventArgs e)

{

HttpApplication app = (HttpApplication)source;

HttpContext context = app.Context;

var auth = DependencyResolver.Current.GetService<IAuthentication>();

**auth.HttpContext = context;**

**context.User = auth.CurrentUser;**

}

public void Dispose()

{

}

}

Вся соль в выделенных строках (**auth.HttpContext = context и context.User = auth.CurrentUser**), как только наш модуль авторизации узнает о контексте и содержащихся в нем кукисах, то моментально получает доступ к имени, по имени он в репозитории получает данные, кто это и возвращает в BaseController. Но не сразу это всё, а по требованию.

Подключаем модуль в Web.config:

<system.web>

…

<httpModules>

<add name="AuthHttpModule" type="LessonProject.Global.Auth.AuthHttpModule"/>

</httpModules>

</system.web>

План таков:

* Наверху показываем авторизован пользователь или (если авторизован, то его email и ссылка на выход, если нет, то ссылки на вход и регистрацию)
* Создаем форму для входа
* Если пользователь правильно ввел данные – то авторизуем его и отправляем на главную страницу
* Если пользователь выходит – то убиваем его авторизацию

Поехали. Добавляем Html.Action(“UserLogin”, “Home”) – это partial view (т.е. кусок кода, который не имеет Layout) – т.е. выводится где прописан, а не в RenderBody().

\_Layout.cshtml:

<body>

**<div class="navbar navbar-fixed-top">**

**<div class="navbar-inner">**

**<div class="container">**

**<ul class="nav nav-pills pull-right">**

**@Html.Action("UserLogin", "Home")**

**</ul>**

**</div>**

**</div>**

**</div>**

@RenderBody()

HomeController.cs:

public ActionResult UserLogin()

{

return View(CurrentUser);

}

UserLogin.cshtml:

@model LessonProject.Model.User

@if (Model != null)

{

<li>@Model.Email</li>

<li>@Html.ActionLink("Выход", "Logout", "Login")</li>

}

else

{

<li>@Html.ActionLink("Вход", "Index", "Login")</li>

<li>@Html.ActionLink("Регистрация", "Register", "User")</li>

}

Контроллер входа выхода LoginController:

public class LoginController : DefaultController

{

[HttpGet]

public ActionResult Index()

{

return View(new LoginView());

}

[HttpPost]

public ActionResult Index(LoginView loginView)

{

if (ModelState.IsValid)

{

var user = Auth.Login(loginView.Email, loginView.Password, loginView.IsPersistent);

if (user != null)

{

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

}

ModelState["Password"].Errors.Add("Пароли не совпадают");

}

return View(loginView);

}

public ActionResult Logout()

{

Auth.LogOut();

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

}

}

LoginView.cs:

public class LoginView

{

[Required(ErrorMessage = "Введите email")]

public string Email { get; set; }

[Required(ErrorMessage = "Введите пароль")]

public string Password { get; set; }

public bool IsPersistent { get; set; }

}

Страница для входа Index.cshtml:

@model LessonProject.Models.ViewModels.LoginView

@{

ViewBag.Title = "Вход";

Layout = "~/Areas/Default/Views/Shared/\_Layout.cshtml";

}

<h2>Вход</h2>

@using (Html.BeginForm("Index", "Login", FormMethod.Post, new { @class = "form-horizontal" }))

{

<fieldset>

<legend>Вход</legend>

<div class="control-group">

<label class="control-label" for="Email">

Email</label>

<div class="controls">

@Html.TextBox("Email", Model.Email, new { @class = "input-xlarge" })

<p class="help-block">Введите Email</p>

@Html.ValidationMessage("Email")

</div>

</div>

<div class="control-group">

<label class="control-label" for="Password">

Пароль</label>

<div class="controls">

@Html.Password("Password", Model.Password, new { @class = "input-xlarge" })

@Html.ValidationMessage("Password")

</div>

</div>

<div class="form-actions">

<button type="submit" class="btn btn-primary">

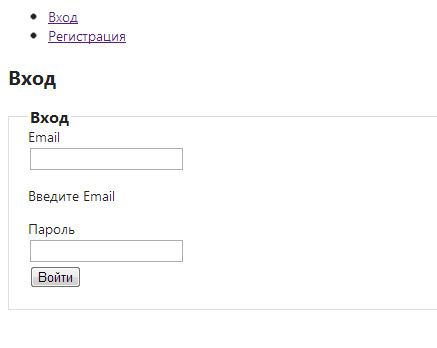
Войти</button>

</div>

</fieldset>

}

Запускаем и проверяем:



После авторизации:

