ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет програмування та комп’ютерних

і телекомунікаційних систем

Кафедра інженерії програмного забезпечення

**Лабораторна робота №**6

з дисципліни «Програмування в інтернет»

на тему:

«Авторизація та аутентифікація в MVC 4. Аутентифікація Windows. Аутентифікація форм. Налаштування використання SimpleMembershipProvider. Використання універсальних провайдерів.»

**Виконав:**

студент 1 курсу, групи ІПЗс-20-1, варіант 8 Н.В.Савич

**Перевірив:**\_\_\_\_\_\_\_\_\_\_\_\_\_\_ О.М.Яшина

(підпис)

Хмельницький – 2020 р.

Мета. Надбати основні поняття розробки механізму авторизації і аутентифікації.

Завдання

Реалізувати авторизацію та аутентифікація користувачів Web - додатка розробленого в завданні до лабораторної роботи №4. Налаштувати SimpleMembershipProvider. Використати універсальні провайдери.

Код Програми:

Моделі:

Type.cs

using System;

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations;

using System.Linq;

using System.Web;

namespace lab1.Models

{

public class Type

{

public int Id { get; set; }

[UIHint("Назва")]

[StringLength(50, MinimumLength = 3, ErrorMessage = "Довжинарядка повинна бути від 3 до 50 символів")]

[Display(Name = "Введіть назву")]

public string Name { get; set; }

[UIHint("Введіть ціну")]

[Display(Name = "Цына")]

[DataType(DataType.Currency)]

public int Price { get; set; }

}

}

Order.cs

using System;

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations;

using System.Linq;

using System.Web;

namespace lab1.Models

{

public class Order

{

[Required]

public int OrderId { get; set; }

[UIHint("ПІБ")]

[StringLength(50, MinimumLength = 3, ErrorMessage = "Довжинарядка повинна бути від 3 до 50 символів")]

[Display(Name = "ПІБ")]

public string Person { get; set; }

[UIHint("Адреса")]

[Display(Name = "Адреса")]

public string Address { get; set; }

public int TypeId { get; set; }

[UIHint("Дата")]

[Display(Name = "Дата")]

public DateTime Date { get; set; }

}

}

AccauntViewModel.cs

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations;

namespace BeekeepersBlog\_V2.Models

{

public class ExternalLoginConfirmationViewModel

{

[Required]

[Display(Name = "Адрес электронной почты")]

public string Email { get; set; }

}

public class ExternalLoginListViewModel

{

public string ReturnUrl { get; set; }

}

public class SendCodeViewModel

{

public string SelectedProvider { get; set; }

public ICollection<System.Web.Mvc.SelectListItem> Providers { get; set; }

public string ReturnUrl { get; set; }

public bool RememberMe { get; set; }

}

public class VerifyCodeViewModel

{

[Required]

public string Provider { get; set; }

[Required]

[Display(Name = "Код")]

public string Code { get; set; }

public string ReturnUrl { get; set; }

[Display(Name = "Запомнить браузер?")]

public bool RememberBrowser { get; set; }

public bool RememberMe { get; set; }

}

public class ForgotViewModel

{

[Required]

[Display(Name = "Адрес электронной почты")]

public string Email { get; set; }

}

public class LoginViewModel

{

[Required]

[Display(Name = "Адрес электронной почты")]

[EmailAddress]

public string Email { get; set; }

[Required]

[DataType(DataType.Password)]

[Display(Name = "Пароль")]

public string Password { get; set; }

[Display(Name = "Запомнить меня")]

public bool RememberMe { get; set; }

}

public class RegisterViewModel

{

[Required]

[EmailAddress]

[Display(Name = "Адрес электронной почты")]

public string Email { get; set; }

[Required]

[StringLength(100, ErrorMessage = "Значение {0} должно содержать не менее {2} символов.", MinimumLength = 6)]

[DataType(DataType.Password)]

[Display(Name = "Пароль")]

public string Password { get; set; }

[DataType(DataType.Password)]

[Display(Name = "Подтверждение пароля")]

[Compare("Password", ErrorMessage = "Пароль и его подтверждение не совпадают.")]

public string ConfirmPassword { get; set; }

}

public class ResetPasswordViewModel

{

[Required]

[EmailAddress]

[Display(Name = "Адрес электронной почты")]

public string Email { get; set; }

[Required]

[StringLength(100, ErrorMessage = "Значение {0} должно содержать не менее {2} символов.", MinimumLength = 6)]

[DataType(DataType.Password)]

[Display(Name = "Пароль")]

public string Password { get; set; }

[DataType(DataType.Password)]

[Display(Name = "Подтверждение пароля")]

[Compare("Password", ErrorMessage = "Пароль и его подтверждение не совпадают.")]

public string ConfirmPassword { get; set; }

public string Code { get; set; }

}

public class ForgotPasswordViewModel

{

[Required]

[EmailAddress]

[Display(Name = "Почта")]

public string Email { get; set; }

}

}

Global.asax:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

using System.Web.Mvc;

using System.Web.Optimization;

using System.Web.Routing;

namespace BeekeepersBlog\_V2

{

public class MvcApplication : System.Web.HttpApplication

{

protected void Application\_Start()

{

AreaRegistration.RegisterAllAreas();

FilterConfig.RegisterGlobalFilters(GlobalFilters.Filters);

RouteConfig.RegisterRoutes(RouteTable.Routes);

BundleConfig.RegisterBundles(BundleTable.Bundles);

}

}

}

Контролери:

AccauntController.cs

[Authorize]

public class AccountController : Controller

{

private ApplicationSignInManager \_signInManager;

private ApplicationUserManager \_userManager;

public AccountController()

{

}

public AccountController(ApplicationUserManager userManager, ApplicationSignInManager signInManager )

{

UserManager = userManager;

SignInManager = signInManager;

}

public ApplicationSignInManager SignInManager

{

get

{

return \_signInManager ?? HttpContext.GetOwinContext().Get<ApplicationSignInManager>();

}

private set

{

\_signInManager = value;

}

}

public ApplicationUserManager UserManager

{

get

{

return \_userManager ?? HttpContext.GetOwinContext().GetUserManager<ApplicationUserManager>();

}

private set

{

\_userManager = value;

}

}

//

// GET: /Account/Login

[AllowAnonymous]

public ActionResult Login(string returnUrl)

{

ViewBag.ReturnUrl = returnUrl;

return View();

}

//

// POST: /Account/Login

[HttpPost]

[AllowAnonymous]

[ValidateAntiForgeryToken]

public async Task<ActionResult> Login(LoginViewModel model, string returnUrl)

{

if (!ModelState.IsValid)

{

return View(model);

}

var result = await SignInManager.PasswordSignInAsync(model.Email, model.Password, model.RememberMe, shouldLockout: false);

switch (result)

{

case SignInStatus.Success:

return RedirectToLocal(returnUrl);

case SignInStatus.LockedOut:

return View("Lockout");

case SignInStatus.RequiresVerification:

return RedirectToAction("SendCode", new { ReturnUrl = returnUrl, RememberMe = model.RememberMe });

case SignInStatus.Failure:

default:

ModelState.AddModelError("", "Неудачная попытка входа.");

return View(model);

}

}

}

HomeController.cs

using lab1.Models;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

using System.Web.Mvc;

namespace lab1.Controllers

{

public class HomeController : Controller

{

TypeContext db = new TypeContext();

public ActionResult Index()

{

IEnumerable<Models.Type> types = db.Types;

ViewBag.Types = types;

return View();

}

[HttpGet]

public ActionResult Buy(int id)

{

ViewBag.BookId = id;

return View();

}

[HttpPost]

public string Buy(Order order)

{

order.Date = DateTime.Now;

db.Orders.Add(order);

db.SaveChanges();

return "Спасибі," + order.Person + ", за покупку!";

}

}

}

TypeController.cs

using System;

using System.Collections.Generic;

using System.Data;

using System.Data.Entity;

using System.Linq;

using System.Net;

using System.Web;

using System.Web.Mvc;

using lab1.Models;

namespace lab1.Controllers

{

public class TypesController : Controller

{

private TypeContext db = new TypeContext();

public ActionResult Index()

{

return View(db.Types.ToList());

}

public ActionResult Create()

{

return View();

}

[HttpPost]

[ValidateAntiForgeryToken]

public ActionResult Create([Bind(Include = "Id,Name,Price")] Models.Type type)

{

if (ModelState.IsValid)

{

db.Types.Add(type);

db.SaveChanges();

return RedirectToAction("Index");

}

return View(type);

}

public ActionResult Edit(int? id)

{

if (id == null)

{

return new HttpStatusCodeResult(HttpStatusCode.BadRequest);

}

Models.Type type = db.Types.Find(id);

if (type == null)

{

return HttpNotFound();

}

return View(type);

}

[HttpPost]

[ValidateAntiForgeryToken]

public ActionResult Edit([Bind(Include = "Id,Name,Price")] Models.Type type)

{

if (ModelState.IsValid)

{

db.Entry(type).State = EntityState.Modified;

db.SaveChanges();

return RedirectToAction("Index");

}

return View(type);

}

public ActionResult Delete(int? id)

{

if (id == null)

{

return new HttpStatusCodeResult(HttpStatusCode.BadRequest);

}

Models.Type type = db.Types.Find(id);

if (type == null)

{

return HttpNotFound();

}

return View(type);

}

[HttpPost, ActionName("Delete")]

[ValidateAntiForgeryToken]

public ActionResult DeleteConfirmed(int id)

{

Models.Type type = db.Types.Find(id);

db.Types.Remove(type);

db.SaveChanges();

return RedirectToAction("Index");

}

protected override void Dispose(bool disposing)

{

if (disposing)

{

db.Dispose();

}

base.Dispose(disposing);

}

}

}

Представлення:

Login.cshtml

@using BeekeepersBlog\_V2.Models

@model LoginViewModel

@{

ViewBag.Title = "Выполнить вход";

}

<h2>@ViewBag.Title.</h2>

<div class="row">

<div class="col-md-8">

<section id="loginForm">

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

{

@Html.AntiForgeryToken()

<h4>Используйте локальную учетную запись для входа.</h4>

<hr />

@Html.ValidationSummary(true, "", new { @class = "text-danger" })

<div class="form-group">

@Html.LabelFor(m => m.Email, new { @class = "col-md-2 control-label" })

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

@Html.TextBoxFor(m => m.Email, new { @class = "form-control" })

@Html.ValidationMessageFor(m => m.Email, "", new { @class = "text-danger" })

</div>

</div>

<div class="form-group">

@Html.LabelFor(m => m.Password, new { @class = "col-md-2 control-label" })

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

@Html.PasswordFor(m => m.Password, new { @class = "form-control" })

@Html.ValidationMessageFor(m => m.Password, "", new { @class = "text-danger" })

</div>

</div>

<div class="form-group">

<div class="col-md-offset-2 col-md-10">

<div class="checkbox">

@Html.CheckBoxFor(m => m.RememberMe)

@Html.LabelFor(m => m.RememberMe)

</div>

</div>

</div>

<div class="form-group">

<div class="col-md-offset-2 col-md-10">

<input type="submit" value="Выполнить вход" class="btn btn-default" />

</div>

</div>

<p>

@Html.ActionLink("Регистрация нового пользователя", "Register")

</p>

}

</section>

</div>

<div class="col-md-4">

<section id="socialLoginForm">

@Html.Partial("\_ExternalLoginsListPartial", new ExternalLoginListViewModel { ReturnUrl = ViewBag.ReturnUrl })

</section>

</div>

</div>

@section Scripts {

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

}

Register.cshtml

@model BeekeepersBlog\_V2.Models.RegisterViewModel

@{

ViewBag.Title = "Регистрация";

}

<h2>@ViewBag.Title.</h2>

@using (Html.BeginForm("Register", "Account", FormMethod.Post, new { @class = "form-horizontal", role = "form" }))

{

@Html.AntiForgeryToken()

<h4>Создайте новую учетную запись.</h4>

<hr />

@Html.ValidationSummary("", new { @class = "text-danger" })

<div class="form-group">

@Html.LabelFor(m => m.Email, new { @class = "col-md-2 control-label" })

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

@Html.TextBoxFor(m => m.Email, new { @class = "form-control" })

</div>

</div>

<div class="form-group">

@Html.LabelFor(m => m.Password, new { @class = "col-md-2 control-label" })

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

@Html.PasswordFor(m => m.Password, new { @class = "form-control" })

</div>

</div>

<div class="form-group">

@Html.LabelFor(m => m.ConfirmPassword, new { @class = "col-md-2 control-label" })

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

@Html.PasswordFor(m => m.ConfirmPassword, new { @class = "form-control" })

</div>

</div>

<div class="form-group">

<div class="col-md-offset-2 col-md-10">

<input type="submit" class="btn btn-default" value="Регистрация" />

</div>

</div>

}

@section Scripts {

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

}

Вигляд програми:



Рис 1.- Вигляд меню головної сторінки не зареєстрованого користувача



Рис 2.- Вигляд меню головної сторінки зареєстрованого користувача

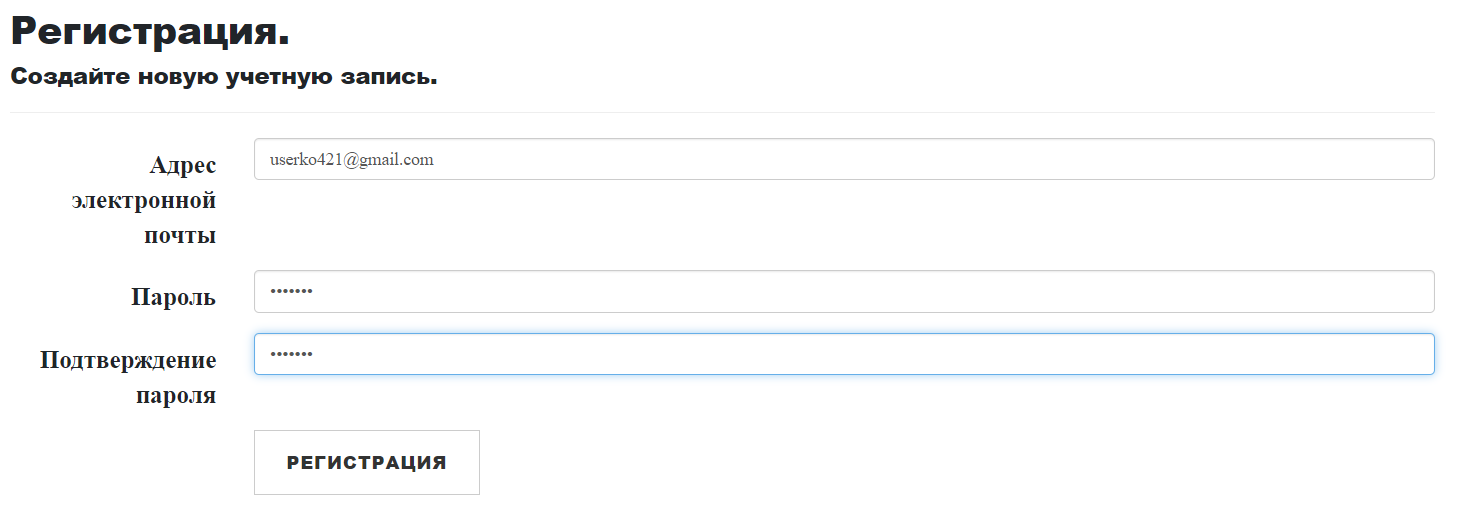


Рис 3. – Вигляд вікна реєстрації

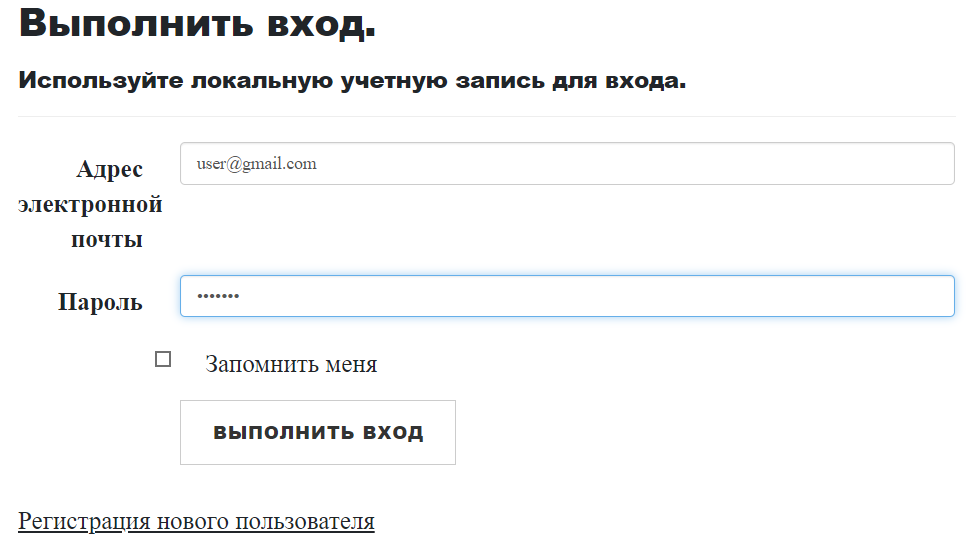


Рис 4. – Вигляд вікна входу

Контрольні питання:

1. Механізми авторизації і аутентифікації

Велику роль у веб-додатках грають механізми авторизації і аутентифікації. Вони дозволяють розмежувати доступ для різних груп користувачів, а також ідентифікувати користувачів. Аутентифікація - це процес ідентифікаціі користувача, тобто ми дізнаємося, що за користувач відвідав веб-додаток. А авторизація вже представляє процес визначення прав, які можуть бути дані аутентифікованим користувачеві, його можливостей по доступу до ресурсів веб-додатка.

2. Авторизація в MVC

Авторизація в MVC опирається на три ключові області, які допомагають керувати користувачами в системі. Це членство (membership), ролі і профілі.

Членство представляє управління реєстраційними даними користувачів, які потім використовуються для авторизації і аутентифікації.

Ролі - групи користувачів, які використовуються для авторизації і розділення доступу до ресурсів.

Профілі представляють різну інформацію про користувачів, як наприклад, ім'я, прізвище, зріст, вік і т.д

3. Управління користувачами

Управління користувачами, членством і ролям проводиться за допомогою провайдерів членства і ролей. У MVC 4 можна використовувати різні провайдери: як у вигляді API SimpleMembershipProvider, так і у вигляді універсальних провайдерів. При необхідності гнучке середовище ASP.NET MVC дозволяє створювати і використовувати свої провайдери членства/ролей і профілів зі своєю унікальною логікою.

4. Провайдер SimpleMembershipProvider

Клас SimpleMembershipProvider покликаний спростити роботу з користувачами і ролями, володіючи 124 більшою гнучкістю і розширюваністю в порівнянні з традиційним провайдером членства. Клас SimpleMembershipProvider також застосовує традиційний провайдер членства: спочатку абстрактний клас ExtendedMembershipProvider наслідує MembershipProvider, а потім сам SimpleMembershipProvider наслідується від ExtendedMembershipProvider.

5. Універсальні провайдери

Інший підхід до авторизації та аутентифікації представляють універсальні провайдери членства і ролей. Хоча в цілому вони надають все той же функціонал, що і SimpleMembershipProvider і SimpleRolesProvider. Шаблон Basic для проекту MVC 4 за замовчуванням вже включає використання універсальних провайдерів.

6. Аутентифікація Windows

Аутентифікація Windows представляє один із способів аутентифікації користувача в системі. При використанні цієї моделі аутентифікації при зверненні користувача до ресурсів веб-додатка разом з HTTP-запитом надсилається і токен безпеки Windows, який і верифікує користувача.

7. Налаштуння додатка на Аутентифікацію Windows

Аутентифікація Windows задана наступним рядком у файлі web.config:

<authentication mode=”Windows” />

Тепер налаштуємо додаток на даний спосіб аутентифікації. Для цього перейдемо до властивостей проекту і встановимо для властивості Anonymous Authentication значення Disabled, а для властивості Windows Authentication значення Enabled.

Тепер вся основна настройка зроблена, і ми можемо увійти в додаток. При запуску нам буде запропоновано ввести логін і пароль.

8. Налаштування авторизації

Застосовуючи фільтри авторизації, ми можемо обмежити доступ до дій контролера або до контролера в цілому для певних ролей (використовуючи групи в Windows) або користувачів:

[Authorize(Users = @"Eugene-ПК\Eugene")]

public ActionResult Index() {

return View();

}

[Authorize(Roles = "Admin, AllUsers")]

public ActionResult About() {

return View();

}

[Authorize(Roles = "Admin")]

public ActionResult Contact() {

return View();

}

9. Аутентифікація форм

Ще один способів аутентифікації користувачів є аутентифікація форм. Вона більш гнучка в порівнянні з аутентификацией Windows, хоча, можливо, і трохи більш складна. Вона грунтується на видачу аутентифікованим користувачеві кукинаборів, за якими він надалі верифікується.

10. Файл конфігурації web.config

Щоб встановити режим аутентифікації на основі форм, у файл конфігурації web.config заноситься наступні рядки в секцію system.web:

<system.web>

……………….

<authentication mode=”Forms”>

<forms loginUrl = “~/Account/Login” timeout=”2880” />

</authentication>

………………….

<system.web>

11. Метод Login контролера AccountController

Метод Login виглядає наступним чином:

[AllowAnonymous]

public ActionResult Login(string returnUrl) {

ViewBag.ReturnUrl = returnUrl;

return View();

}

//

// POST: /Account/Login

[HttpPost]

[AllowAnonymous]

[ValidateAntiForgeryToken]

public ActionResult Login(LoginModel model, string returnUrl) {

if (ModelState.IsValid && WebSecurity.Login(model.UserName, model.Password, persistCookie: model.RememberMe)) {

return RedirectToLocal(returnUrl);

}

// If we got this far, something failed, redisplay form

ModelState.AddModelError("", "The user name or password provided is incorrect.");

return View(model);

}

12. Валідація користувача

Валідація користувача: це робить метод Membership.ValidateUser. І якщо користувач знаходиться в нашій базі даних, то далі додаток переходить до другого етапу

13. Аутентифікаційний тікет

Другий етап являє створення аутентифікаційного тікета - тобто деякого квитка безпеки, за яким веб-додаток буде впізнавати користувача. Цей тікет додаток встановлює для браузера у вигляді куки-набору на ім'я .AUTHPATH за допомогою методу FormsAuthentication.SetAuthCookie.

14. Створення куки-набору

Важливо відзначити, що при установці куки-наборів їх вміст шифрується за допомогою машинних ключів, яку автоматично створює IIS на сервері.

15. Налаштування аутентифікації

Використовуючи атрибути вузла forms у файлі конфігурації, ми можемо налаштувати параметри аутентифікації. Зокрема ми можемо застосувати наступні параметри:

- coockieless: визначає, чи застосовуються куки-набори і як вони використовуються. Може приймати такі значення: UseCookies (визначає, що куки-набори будуть використовуватися завжди незалежно від пристрою), UseUri (куки-набори ніколи не використовуються), AutoDetect (якщо пристрій підтримує кукинабори, то вони використовуються, в іншому випадку вони не застосовуються, при цьому проводиться тестування, що визначає, чи включена підтримка), UseDeviceProfile (використовується за замовчуванням) (якщо пристрій підтримує куки-набори, то вони використовуються, в іншому випадку вони не застосовуються, на відміну від попереднього випадку тестування не проводиться);

- defaultUrl: визначає шлях, по якому здійснюється перехід після авторизації;

- domain: визначає куки-набори для всього домену. Завдяки цьому ми можемо використовувати одні й ті ж куки-набра для головного домену та його субдоменів. За замовчуванням має значення як порожнього рядка;

- loginUrl: адреса для аутентифікації користувача. Значення за замовчуванням - "~ / Account / Login";

- name: задає ім'я для куки-набору. Значення за замовчуванням - ".ASPXAUTH";

- path: шлях для куки-наборів. Значення за замовчуванням - "/"

- requireSSL: визначає, чи SSL-з'єднання для передачі кукинаборів. Значення за замовчуванням false;

- timeout: визначає термін дії куків у хвилинах;

16. Таблиці за замовчуванням в ASP.NET MVC 4

При першому зверненні до БД, якщо її не існувало, вони будуть створені і автоматично заповнені таблицями з певними полями

Такі таблиці як webpages\_Membership, webpages\_OAuthMembership, webpages\_Roles, webpages\_UsersInRoles.