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 12 Д.А.Ямборко

**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:

Файл конфигурации.

Configuration.cs

protected override void Seed(LR5\_1.Models.ApplicationDbContext context)

{

context.Phones.Add(new Phone { Mark = "Phone11",Model="A20", Price = 12000});

// This method will be called after migrating to the latest version.

// You can use the DbSet<T>.AddOrUpdate() helper extension method

// to avoid creating duplicate seed data.

var userManager = new ApplicationUserManager(new UserStore<ApplicationUser>(context));

var roleManager = new RoleManager<IdentityRole>(new RoleStore<IdentityRole>(context));

var role1 = new IdentityRole { Name = "admin" };

var role2 = new IdentityRole { Name = "user" };

roleManager.Create(role1);

roleManager.Create(role2);

var admin = new ApplicationUser { Email = "DimaIamborko@mail.ru", UserName = "DimaIamborko@mail.ru" };

string password = "Dima\_1234";

string password = "Dgelsomin1\_";

var result = userManager.Create(admin, password);

if (result.Succeeded)

{

userManager.AddToRole(admin.Id, role1.Name);

userManager.AddToRole(admin.Id, role2.Name);

}

base.Seed(context);

}

Контролери:

AccountController.cs

[HttpPost]

[AllowAnonymous]

[ValidateAntiForgeryToken]

public async Task<ActionResult> Register(RegisterViewModel model)

{

if (ModelState.IsValid)

{

var user = new ApplicationUser { UserName = model.Email, Email = model.Email, Name = model.Name, Surname = model.Surname };

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

if (result.Succeeded)

{

await SignInManager.SignInAsync(user, isPersistent:false, rememberBrowser:false);

// Дополнительные сведения о включении подтверждения учетной записи и сброса пароля см. на странице https://go.microsoft.com/fwlink/?LinkID=320771.

// Отправка сообщения электронной почты с этой ссылкой

// string code = await UserManager.GenerateEmailConfirmationTokenAsync(user.Id);

// var callbackUrl = Url.Action("ConfirmEmail", "Account", new { userId = user.Id, code = code }, protocol: Request.Url.Scheme);

// await UserManager.SendEmailAsync(user.Id, "Подтверждение учетной записи", "Подтвердите вашу учетную запись, щелкнув <a href=\"" + callbackUrl + "\">здесь</a>");

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

}

AddErrors(result);

}

// Появление этого сообщения означает наличие ошибки; повторное отображение формы

return View(model);

}

Представлення:

\_Layout.cshtml

<!DOCTYPE html>

<html>

<head>

<meta http-equiv="Content-Type" content="text/html; charset=utf-8"/>

<meta charset="utf-8" />

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>@ViewBag.Title - приложение ASP.NET</title>

@Styles.Render("~/Content/css")

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

</head>

<body>

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

<div class="container">

<div class="navbar-header">

<button type="button" class="navbar-toggle" data-toggle="collapse" data-target=".navbar-collapse">

<span class="icon-bar"></span>

<span class="icon-bar"></span>

<span class="icon-bar"></span>

</button>

@Html.ActionLink("Имя приложения", "Index", "Home", new { area = "" }, new { @class = "navbar-brand" })

</div>

<div class="navbar-collapse collapse">

<ul class="nav navbar-nav">

<li>@Html.ActionLink("Домашняя страница", "Index", "Home")</li>

@if (User.Identity.IsAuthenticated && User.IsInRole("admin")) {

<li>@Html.ActionLink("Информация", "About", "Home")</li>}

<li>@Html.ActionLink("Связаться", "Contact", "Home")</li>}

</ul>

@Html.Partial("\_LoginPartial")

</div>

</div>

</div>

<div class="container body-content">

@RenderBody()

<hr />

<footer>

<p>&copy; @DateTime.Now.Year - мое приложение ASP.NET</p>

</footer>

</div>

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

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

@RenderSection("scripts", required: false)

</body>

</html>

Register.cshtml

@model Lr1.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>Создание новой учетной записи.</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.Name, new { @class = "col-md-2 control-label" })

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

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

</div>

</div>

<div class="form-group">

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

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

@Html.PasswordFor(m => m.Surname, 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>

}

Вигляд програми:

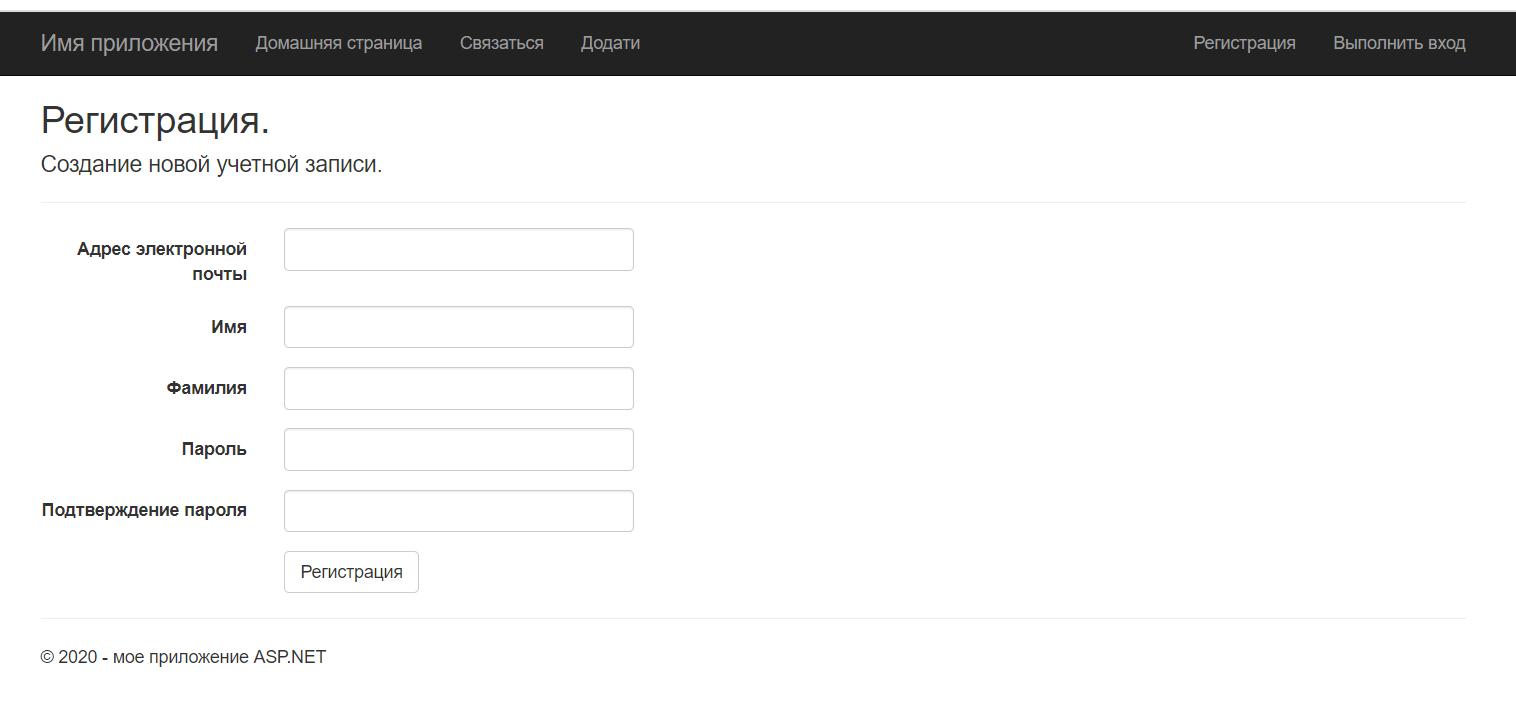


Рис 1.- Вигляд форми реєстрації

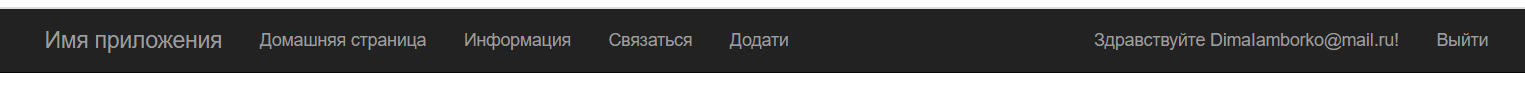


Рис 2. – Вигляд шапки сайту користувача зареэстрованого як адмін

GIT repository: https://github.com/alsowentafk/1\_semestr\_knu/tree/master/PI/ASP\_DIma-1/LR-4-6/Lr1

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.