УДК 004

АВТОРИЗАЦИЯ И АУТЕНТИФИКАЦИЯ ПОЛЬЗОВАТЕЛЕЙ В ВЕБ-ПРИЛОЖЕНИЯХ НА ОСНОВЕ ASP.NET CORE IDENTITY

Юркин С.Ю., студент гр. ИТб-131, IV курс
Научный руководитель: Турчин Д.Е., старший преподаватель
Кузбасский государственный технический университет
имени Т.Ф. Горбачева
г. Кемерово

На сегодняшний день практически ни одно веб-приложение не обходится без системы авторизации и аутентификации. В данной статье будет рассмотрена реализация авторизации и аутентификации пользователей в ASP.NET Core приложениях на основе системы ASP.NET Core Identity.

Платформа ASP.NET представляет технологию от компании Microsoft, предназначенную для создания различного рода веб-приложений от небольших веб-сайтов до крупных веб-порталов и веб-сервисов.

В 2016 году компания Microsoft представила новую версию ASP.NET платформы - ASP.NET Core. Релиз ознаменовался выходом новой системы авторизации и аутентификации – ASP.NET Core Identity. Эта система позволяет пользователям аутентифицироваться, создавать учетные записи, управлять ими или использовать для входа на сайт учетные записи внешних провайдеров, таких как Google, Facebook, Microsoft, Twitter и других. Следует отметить, что ASP.NET Core теперь является полностью opensource-фреймворком и построен на основе кроссплатформенной среды .NET Core, которая может быть развернута на основных популярных операционных системах: Windows, Mac OS X и Linux.

Основные пакеты Asp.Net Core Identity

Основными пакетами системы являются:

– Microsoft.AspNetCore.Identity - основная библиотека ASP.NET Identity, которая хранит весь базовый функционал;

– Microsoft.AspNetCore.Identity.EntityFrameworkCore - реализация интерфейсов ASP.NET Identity с использованием Entity Framework, которая позволяет взаимодействовать с хранилищем данных;

– Microsoft.AspNetCore.Authentication.Cookies - пакет, позволяющий использовать в приложении аутентификацию на основе куки;

– Microsoft.AspNetCore.Cryptography.KeyDerivation - функциональность для работы с ключами безопасности и шифрованием;

– Microsoft.AspNetCore.Hosting.Abstractions - абстракции для хостирования приложения в ASP.NET.

Контекст данных

Для работы с базой данных используется контекст данных, который представляет собой простой класс, наследуемый от класса IdentityDbContext пространства имен Microsoft.AspNetCore.Identity.EntityFrameworkCore. Контекст данных позволяет получить содержимое таблиц из базы данных. по умолчанию для ASP.NET Identity это следующие таблицы:

– Users - набор IdentityUser, соответствует таблице пользователей;

– Roles - набор объектов IdentityRole, соответствует таблице ролей;

– RoleClaims - набор объектов IdentityRoleClaim, соответствует таблице связи ролей и объектов claims;

– UserLogins - набор объектов IdentityUserLogin, соответствует таблице связи пользователей с их логинами их внешних сервисов;

– UserClaims - набор объектов IdentityUserClaim, соответствует таблице связи пользователей и объектов claims;

– UserRoles - набор объектов IdentityUserRole, соответствует таблице, которая сопоставляет пользователей и их роли;

– UserTokens - набор объектов IdentityUserToken, соответствует таблице токенов пользователей.

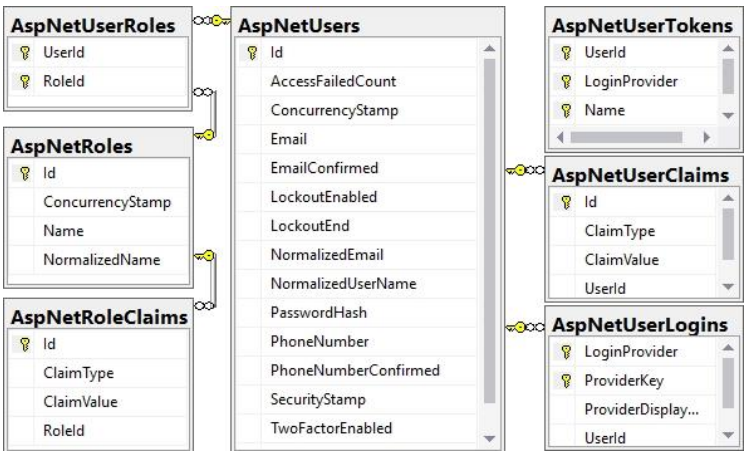


Рисунок 1 – Диаграмма базы данных Identity

На рисунке 1 представлена диаграмма базы данных, используемая ASP.NET Core Identity по умолчанию.

Если нам надо хранить в базе данных объекты каких-то других классов, то в классе контекста можно определить для них свойство по типу DbSet<T>.

Представление пользователей

В ASP.NET Core Identity пользователь представлен классом IdentityUser из пространства имен Microsoft.AspNetCore.Identity.EntityFrameworkCore. Этот класс предоставляет базовую информацию о пользователе. Его основные свойства:

– Id - уникальный идентификатор пользователя;

– UserName - ник пользователя;

– Email – адрес эл. почты пользователя;

– PasswordHash - хеш пароля. В базе данных напрямую не хранится пароль, а только его хеш;

– Roles - набор ролей, к которым принадлежит пользователь;

Менеджер пользователей

Для управления пользователями используется не контекст данных, а специальный класс UserManager<T> из пространства имен Microsoft.AspNetCore.Identity. Основные из его методов и свойств – это изменение пароля, создание нового пользователя, удаление пользователя, поиск пользователя и др.

Авторизация пользователей

Рассмотрим процесс авторизации пользователя по адресу эл. почты и паролю. Допустим, в системе есть зарегистрированный пользователь, для которого установлены email: example@mail.ru и password: Secret1234$. Запись о нем в базе данных представлена на рисунке 2.

	Id	Email	PasswordHash	UserName
1	3272aff6-d636-4...	example@mail.ru	AQAAAAEAACcQAAAAEONjmbk6eweZOgPFYkf...	example@mail.ru

Рисунок 2. Запись о пользователе в таблице Users

Как видно из рисунка 2, пароль пользователя хранится в зашифрованном виде.

Для формирования html-страницы и передачи данных на сервер используется класс LoginViewModel, представляющий Модель в структуре MVC (Model-View-Controller) приложения и содержащий два параметра: адрес эл. почты и пароль:

```
public class LoginViewModel
{
    public string Email { get; set; }
    public string Password { get; set; }
}
```

Его главная задача – описание структуры и логики используемых данных. Данные поступают на сервер и передаются для обработки в POST-версию метода указанного контроллера. В данном случае это AccountController.Login(). Реализация метода представлена на рисунке 3.

```
[HttpPost]
public async Task<IActionResult> Login(LoginViewModel model, string returnUrl = null)
{
    ViewData["ReturnUrl"] = returnUrl;
    if (ModelState.IsValid)
    {
        var result = await _signInManager.PasswordSignInAsync(
            model.Email, model.Password, false, lockoutOnFailure: false);
        if (result.Succeeded) return RedirectToLocal(returnUrl);
        else
        {
            ModelState.AddModelError("", "Не верный адрес эл. почты или (и) пароль!");
            return View(model);
        }
    }
    return View(model);
}
```

Рисунок 3 – Реализация метода AccountController.Login()

Метод Login получает данные из представления в виде модели LoginViewModel. Всю работу по аутентификации пользователя выполняет метод signInManager.PasswordSignInAsync(). Этот метод принимает логин и пароль пользователя и возвращает экземпляр класса IdentityResult, с помощью которого можно узнать, завершилась ли аутентификация успешно. Если она завершилась успешно, то используется свойство ReturnUrl модели LoginViewModel для возврата пользователя на предыдущую страницу. В противном случае, пользователь возвращается на страницу авторизации и выводится сообщение об ошибке.

Разграничение доступа по ролям

Для разграничения доступа к различным частям приложения в ASP.NET Core Identity применяются роли, представленные классом IdentityRole, который определяет три свойства:

– Id - идентификатор роли;

– Name - название роли;

– Users - коллекция объектов IdentityUserRole, через которые пользователи ассоциированы с данной ролью. Пользователь может иметь одну или несколько ролей или не иметь их вовсе.

По умолчанию в базе данных роли хранятся в таблице Roles и связаны с таблицей Users отношением многие-ко-многим при помощи таблицы UserRoles (рисунок 1).

Допустим, в системе уже установлены две роли: “User” и “Admin”, представляющие пользователя и администратора соответственно. Запись о ролях представлена на рисунке 4, а их отношения с пользователями представлены на рисунке 5.

	Id	Name	NormalizedName
1	7e33c4c5-db69-41e7-8bc7-4f751bd88187	User	USER
2	e6cd5f39-3cb2-4ace-8f96-1ccb4fb98faf	Admin	ADMIN

Рисунок 4 – Запись в таблице Roles

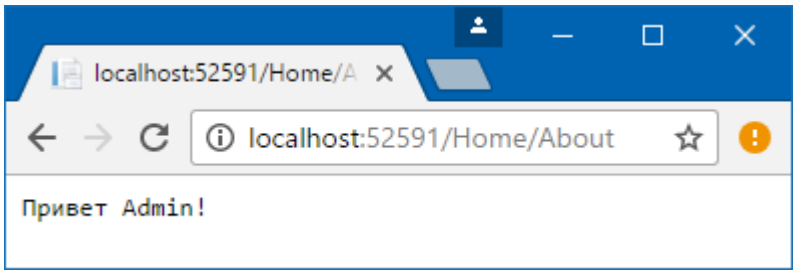
	UserId	RoleId
1	645b7893-e8b5-4b92-89c5-11fb59e87ee5	e6cd5f39-3cb2-4ace-8f96-1ccb4fb98faf

Рисунок 5 – Запись в таблице UserRoles

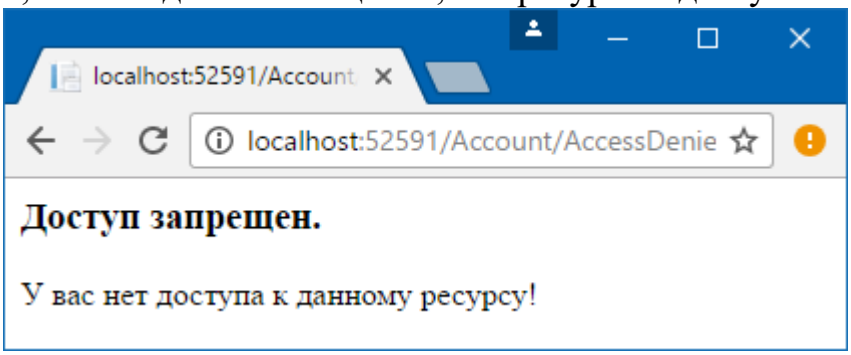
Чтобы обеспечить разграничение по ролям, необходимо контроллеру или его действиям установить атрибут Authorize. Например, контроллер HomeController содержит действие About, которое возвращает в браузер простую строку:

```
[Authorize(Roles = "Admin")]
public string About() => "Привет Admin!";
```

Здесь указано, что к действию About имеют доступ только пользователи, принадлежащие к роли “Admin”. Если мы зайдем в систему под пользователем с правами администратора, и перейдем на странницу …/Home/About, то увидим сообщение “Привет Admin!”:



Но если авторизованный пользователь не будет обладать правами администратора, то выводится сообщение, что ресурс не доступен:



Заключение

В рамках данной статьи была рассмотрена лишь малая часть возможностей технологии ASP.NET Core Identity.

Можно выделить следующие основные преимущества фреймворка:

– кроссплатформенность - возможность использования в приложениях ASP.NET Core на Windows, Mac и Linux;

– расширяемость - возможность дополнять функционал фреймворка под определенные нужды;

– легковесность фреймворка;

– распространение и обновление пакетов через сервис NuGet;

– возможность внедрения в любой тип проекта ASP.NET Core (MVC, WebForms, WebPages, WepAPI и SignalR);

– простая интеграция с социальными сервисами и многое другое.

Учитывая возможности и достоинства данного фреймворка, можно с уверенностью сказать, что ASP.NET Core Identity – это большой скачок в развитии веб-программирования на ASP.NET.

Список литературы:

1. Adam Freeman. Pro ASP.NET Core MVC: Sixth Edition / Adam Freeman – Apress, 2016. – 1018 с.

2. Introduction to ASP.NET Core [Электронный ресурс]: официальная документация для ASP.NET Core. URL: https://docs.microsoft.com/en-us/aspnet/core/.

3. GitHub - ASP.NET Core Identity [Электронный ресурс]: открытый исходный код и документация для ASP.NET Core Identity. URL: https://github.com/aspnet/Identity#aspnet-core-identity.