Куший Д.Н., Федотенко И.И.

Сравнение и примеры использование средств защиты информации в приложениях ASP.NET CORE

*Южно-Российский государственный политехнический университет (НПИ)
имени М.И. Платова
(Россия, Новочеркасск)*

doi:10.18411/sciencepublic-12-08-2020-03

idsp: sciencepublic-12-08-2020-03

Аннотация

В статье рассмотрены компоненты платформы ASP.NET Core, используемые для обеспечения безопасности. Указаны пространства имен, необходимые для аутентификации и авторизации, даны краткие примеры использования методов соответствующих классов. Рассмотрена структура пространства имен System.Security.Cryptography, определены уровни классов шифрования и дешифрования информации.

Ключевые слова: Авторизация, аутентификация, веб-приложения, криптография, ASP.NET Core

Платформа ASP.NET Core представляет собой модульную технологию разработки программного обеспечения с открытым исходным кодом. Несмотря на то, что ASP.NET Core уступает по популярности использованию при разработке веб-ресурсов серверному языку программирования PHP, программисты .NET востребованы в IT-сфере, о чем говорит большое количество размещенных в сети высокооплачиваемых вакансий.

Данный стек технологий поддерживает следующие компоненты обеспечения безопасности в веб-приложениях:

- система идентификации, аутентификации и разграничения доступа ASP.NET Identity;
- пространство имен System.Security.Cryptography с классами для шифрования и дешифрования информации.

При создании проекта в интегрированной среде разработки Visual Studio на базе ASP.NET Core (рисунок 1) можно выбрать один из следующих типов аутентификации:

- без проверки подлинности (No Authentication) – ASP.NET Identity и встроенная система аутентификации отсутствует;
- учетные записи отдельных пользователей (Individual User Accounts) – проект по умолчанию включает систему ASP.NET Identity, которая позволяет авторизовать пользователей как внутри приложения, так и с помощью внешних сервисов;
- рабочие или учебные учетные записи (Organizational Account) – данный тип аутентификации применяется для сайтов и веб-приложений отдельных компаний и организаций;
- аутентификация Windows (Windows Authentication) – система аутентификации для сетей intranet с помощью учетных записей операционной системы Windows.

В большинстве случаев при создании веб-приложений на базе ASP.NET Core используется тип аутентификации на основе учетных записей пользователей с сохранением необходимых данных в локальной базе данных [1]. Конечный набор функций ASP.NET Identity настраивается для каждого конкретного проекта путем добавления требуемых шаблонов.

При использовании базы данных, разработанной с помощью Entity Framework, ее взаимодействие с приложением осуществляется при помощи механизма миграции, запускаемого в консоли пакетов Package Manager Console.

Рисунок 1 – Настройка проверки подлинности

В программной реализации процедур используются свойства и методы классов IdentityUser из пространства имен Microsoft.AspNetCore.Identity.EntityFrameworkCore и UserManager<T>, определенного в Microsoft.AspNetCore.Identity.

Для каждого веб-приложения создается класс модели, описывающей пользователя, с наследованием от IdentityUser [2]:

```
public class User : IdentityUser<int, CustomRole, CustomClaim>
{
    [Key]
    public int Id_user { get; set; }
    public string Login { get; set; }
    public string Password { get; set; }
}
```

```
public string Surname { get; set; }
public string Name { get; set; }
public string Patronymic { get; set; }
public string E-mail { get; set; }
public Role Role { get; set; }
}
```

В представленной реализации идентификатор имеет тип `int` универсального параметра `TKey`.

Поскольку тип аутентификации должен совпадать с идентификатором, определенным в `CookieAuthenticationOptions.AuthenticationType` переопределяется также часть методов:

```
public async Task<ClaimsIdentity> GenerateUserIdentityAsync(
    UserManager<ApplicationUser> manager)
{
    var userIdentity = await manager.CreateIdentityAsync(
        this, DefaultAuthenticationTypes.ApplicationCookie);
    return userIdentity;
}

public class Role : IdentityRole<int, CustomRole>
{
    public Role() { }
    public Role(string Title) { Name = Title; }
}

public class CustomClaim : IdentityUserClaim<int> { }
```

Также изменения вносятся в метод описания контекста, используемого при работе с базой данной:

```
public class ApplicationDbContext : IdentityDbContext<User, Role,
    int, CustomUserLogin, CustomUserRole, CustomUserClaim>
{ ... }
```

В процессе построения решения проекта могут возникнуть ошибки в классах инфраструктуры, используемых в ASP.NET Identity. Класс идентификатора ожидает целое число для ключа, но контроллера (или веб-формы) передается строковое значение. В каждом случае необходимо преобразовать в строку и целое число путем вызова `GetIdUser<int>`.

Если ограничить доступ к контроллеру администратора – будет полученная «классическая проблема» – администратор управляет пользователями и ролями, но нельзя создать первого администратора, так доступ к этим методам ограничен.

Решением этой проблемы является заполнение базы данных первоначальными данными при ее создании средствами Entity Framework. Это позволит автоматически создать пользователей и назначить им роли при первом развертывании приложения. Для этого необходимо использовать метод `PerformInitialSetup` класса `IdentityDbInit`, являющегося специфичным для Entity Framework и ASP.NET Identity, что позволит использовать атрибут `Authorize` для защиты контроллеров администратора, как показано в примерах ниже:

```
[Authorize(Roles = "Administrators")]
public class AdminController : Controller
{ // ... }

[Authorize(Roles = "Administrators")]
public class RoleAdminController : Controller
{ // ... }
```

Использование такой иерархии классов позволяет добавлять новые функциональные и опциональные возможности к существующему набору.

Контроллеры AccountController и ManageController поддерживают функциональные возможности системы.

Следует также отметить, что платформа ASP.NET Core имеет встроенную поддержку аутентификации на основе cookie, реализуемую специальным компонентом middleware [3] пространства имен System.Web.HttpCookie. С его помощью происходит сериализация личных данных пользователя в зашифрованные аутентификационные связи:

```
services.AddAuthentication(  
    CookieAuthenticationDefaults.AuthenticationScheme)  
    .AddCookie(options =>  
    {  
        options.LoginPath =  
            new Microsoft.AspNetCore.Http.PathString("/Account/Login");  
    });
```

При получении запроса от клиента, в котором содержатся cookie, происходит их валидация, десериализация и инициализация свойства User объекта HttpContext.

В тех случаях, когда клиентом cookie не поддерживаются, можно использовать Json Web Token (JWT) – способ безопасной передачи данных в формате JSON, описанный в стандарте RFC 7519 [4]. В веб-приложениях JWT, как правило, передается в заголовке каждого запроса и в стандартном виде имеет следующую структуру: HEADER.PAYLOAD.SIGNATURE.

При использовании зашифрованных данных JSON объект имеет вид: HEADER.ENCRYPTEDKEY.INITVECTOR.CIPHERTEXT.AUTHTAG. Блоки ENCRYPTEDKEY, INITVECTOR и AUTHTAG зависят от используемого алгоритма шифрования, а также могут быть представлены пустой последовательностью.

Выбор того или иного способа зависит от задач, поставленных перед разработчиком. В случае проектирования и создания системы для крупного предприятия с множеством модулей, взаимодействующих друг с другом с помощью сложных логических схем и в различных режимах (синхронном, асинхронном), а также глубокой политикой безопасности выбор стоит остановить на ASP.NET Identity. Объектная ориентированность данной технологии позволит упростить процедуру взаимодействия между различными модулями. В тоже время встроенный набор функций позволяет быстро и качественно создать гибкое и функциональное решение.

Однако стоит отметить, что использование .NET Identity с проведением дополнительных модификаций в инфраструктуре проекта у начинающего .NET Core разработчика может вызвать затруднения за счет сложной архитектуры самой технологии. Кроме того, следует отметить высокую зависимость между программной реализацией и структурой хранимых данных.

В основе авторизации в ASP.NET Core MVC лежит понятия «claim», «identity» и «principal». Первое из них обозначает некоторое свойство пользователя. Все предопределенные характеристики находятся в System.Security.Claims.ClaimTypes, однако данный список может быть изменен разработчиком. Набор свойств, связанных в виде единого выражения, называется «identity», и для работы с ним используется System.Security.Claims.ClaimsIdentity. Кроме того, у одного пользователя может быть несколько «identity», объединенных высокоуровневым понятием «principal» и описываемых типом System.Security.Claims.ClaimsPrincipal.

Однако начинающим веб-разработчикам будет сложно выполнить настройку встроенных шаблонов из-за сложной архитектуры данного решения

Дополнительная защита информации в ASP.NET может быть обеспечена пространством имен System.Security.Cryptography, содержащим следующие категории классов шифрования и дешифрования информации [5]:

- алгоритмы шифрования, в том числе, хеширования и создания цифровых подписей;
- вспомогательные классы для создания криптографически строгих случайных чисел;
- сертификаты X509;
- сигнатуры и шифрование XML в соответствии со стандартами, опубликованными консорциумом W3C.

Следует также отметить, что классы криптографии в .NET делятся на три уровня. На первом уровне абстрактными классами (*AsymmetricAlgorithm*, *SymmetricAlgorithm* и *HashAlgorithm*) определяется общая задача шифрования. Классы второго уровня также являются абстрактными и представляют специфические алгоритмы шифрования. Классы третьего уровня представляют набор реализаций шифрования.

Корпорация Microsoft уделяет большое внимание безопасности своих веб-сервисов, что находит отражение не только в открытой в 2014 году бессрочной Bug Bounty программе, но и в наличии методов встроенной защиты от взломов приложений [6]. ASP.Net Core содержит ряд механизмов для предотвращения таких атак на веб-системы, как SQL инъекции, внедрение в выдаваемую приложением страницу вредоносного кода (XSS), перенаправление пользователя на атакующий сайт (Open redirect attacks), добавление скрытой кнопки/ссылки внутри прозрачного iframe (Click-Jacking) и другие виды взломов.

Комбинация базовых методов аутентификации, защиты информации ASP.Net Core и дополнительной настройки системы безопасности под конкретный проект позволит создать надежную веб-систему.

1. Гринченков Д.В., Куций Д.Н., Фукс А.А. Использование платформы ASP.NET IDENTITY для реализации средств аутентификации и авторизации пользователей в веб-приложениях // Фундаментально-прикладные проблемы безопасности, живучести, надежности, устойчивости и эффективности систем: материалы III междунар. научно-практ. конф., посвященной 110-летию со дня рождения академика Н.А. Пилюгина, 3-5 июня 2019 г. Часть I. – Елец, Елецкий государственный университет им И.А. Бунина, 2019. – С.113-118.
2. Куций Д.Н. К вопросу разработки физической модели данных системы тематического поиска электронных образовательных ресурсов в сети Интернет // Информационно-телекоммуникационные системы и технологии (ИТСиТ-2018): материалы Всерос. науч.-практ. конф., г. Кемерово, 11-13 окт. 2018 г. / Кузбас. гос. техн. ун-т им. Т.Ф. Горбачева - Кемерово, 2018. – С.59-61.
3. Аутентификация и авторизация. Сайт <https://metanit.com>. [Электронный ресурс]. Дата обновления 04.11.2018. URL: <https://metanit.com/sharp/aspnet5/15.1.php> (дата обращения: 01.06.2020).
4. ASP.NET Core: Аутентификация API. Часть 1: JWT. Сайт <https://andrey.moveax.ru>. [Электронный ресурс]. Дата обновления 04.11.2018. URL: <https://andrey.moveax.ru/post/asp-net-core-web-api-authentication-part-1-jwt> (дата обращения: 20.05.2020).
5. Криптография в ASP.NET. Сайт <https://professorweb.ru>. [Электронный ресурс]. URL: https://professorweb.ru/my/ASP_NET/security/level5/5_1.php (дата обращения: 15.05.2020).
6. ASP.NET Core: Механизмы предотвращения атак 2.0. Сайт <https://habr.com>. [Электронный ресурс]. Дата обновления 28.02.2018. URL: <https://habr.com/ru/company/jugru/blog/350188/> (дата обращения: 10.01.2020).