Аутентификация с помощью JWT-токенов

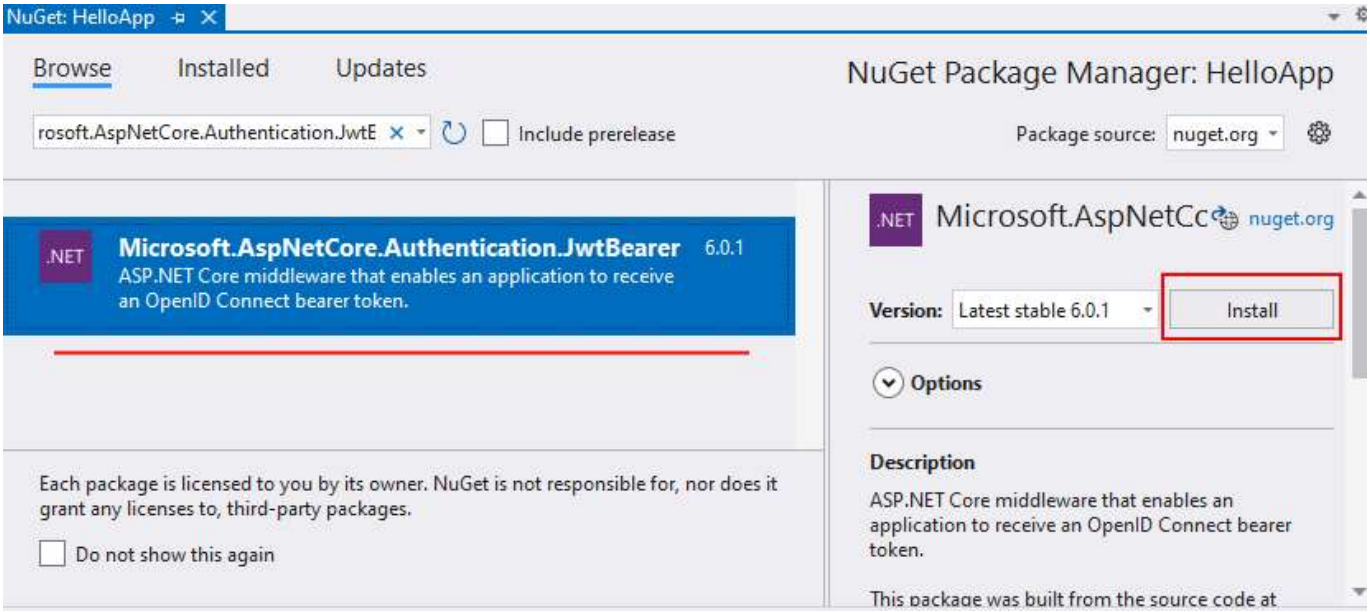
Последнее обновление: 09.01.2022

Одним из подходов к авторизации и аутентификации в ASP.NET Core представляет механизм аутентификации и авторизации с помощью JWT-токенов. Что такое JWT-токен? JWT (или JSON Web Token) представляет собой веб-стандарт, который определяет способ передачи данных о пользователе в формате JSON в зашифрованном виде.

JWT-токен состоит из трех частей:

- **Header** - объект JSON, который содержит информацию о типе токена и алгоритме его шифрования
- **Payload** - объект JSON, который содержит данные, нужные для авторизации пользователя
- **Signature** - строка, которая создается с помощью секретного кода, Headera и Payload. Эта строка служит для верификации токена

Для использования JWT-токенов в проект ASP.NET Core необходимо добавить Nuget-пакет **Microsoft.AspNetCore.Authentication.JwtBearer**.



Сначала рассмотрим принцип генерации и отправки jwt-токена. Для этого в файле **Program.cs** определим следующий код приложения:

```
using Microsoft.AspNetCore.Authentication.JwtBearer;
using Microsoft.AspNetCore.Authorization;
using Microsoft.IdentityModel.Tokens;
using System.IdentityModel.Tokens.Jwt;
using System.Security.Claims;
using System.Text;

var builder = WebApplication.CreateBuilder();

builder.Services.AddAuthorization();
builder.Services.AddAuthentication(JwtBearerDefaults.AuthenticationScheme)
    .AddJwtBearer(options =>
    {
        options.TokenValidationParameters = new TokenValidationParameters
        {
            // указывает, будет ли валидироваться издатель при валидации токена
            ValidateIssuer = true,
            // строка, представляющая издателя
            ValidIssuer = AuthOptions.ISSUER,
            // будет ли валидироваться потребитель токена
            ValidateAudience = true,
            // установка потребителя токена
            ValidAudience = AuthOptions.AUDIENCE,
            // будет ли валидироваться время существования
            ValidateLifetime = true,
            // установка ключа безопасности
            IssuerSigningKey = AuthOptions.GetSymmetricSecurityKey(),
            // валидация ключа безопасности
            ValidateIssuerSigningKey = true,
         };
});
var app = builder.Build();

app.UseAuthentication();
app.UseAuthorization();

app.Map("/login/{username}", (string username) =>
{
    var claims = new List<Claim> {new Claim(ClaimTypes.Name, username) };
    // создаем JWT-токен
    var jwt = new JwtSecurityToken(
            issuer: AuthOptions.ISSUER,
            audience: AuthOptions.AUDIENCE,
            claims: claims,
            expires: DateTime.UtcNow.Add(TimeSpan.FromMinutes(2)),
            signingCredentials: new SigningCredentials(AuthOptions.GetSymmetricSecu
```

```
47
48      return new JwtSecurityTokenHandler().WriteToken(jwt);
49  });
50
51  app.Map("/data", [Authorize] () => new { message= "Hello World!" });
52
53  app.Run();
54
55  public class AuthOptions
56  {
57      public const string ISSUER = "MyAuthServer"; // издатель токена
58      public const string AUDIENCE = "MyAuthClient"; // потребитель токена
59      const string KEY = "mysupersecret_secretkey!123";   // ключ для шифрации
60      public static SymmetricSecurityKey GetSymmetricSecurityKey() =>
61          new SymmetricSecurityKey(Encoding.UTF8.GetBytes(KEY));
62  }
```

Для описания некоторых настроек генерации токена в конце кода определен специальный класс **AuthOptions**:

```
1  public class AuthOptions
2  {
3      public const string ISSUER = "MyAuthServer"; // издатель токена
4      public const string AUDIENCE = "MyAuthClient"; // потребитель токена
5      const string KEY = "mysupersecret_secretkey!123";   // ключ для шифрации
6      public static SymmetricSecurityKey GetSymmetricSecurityKey() =>
7          new SymmetricSecurityKey(Encoding.UTF8.GetBytes(KEY));
8  }
```

Константа `ISSUER` представляет издателя токена. Здесь можно определить любое название.

Константа `AUDIENCE` представляет потребителя токена - опять же может быть любая строка, обычно это сайт, на котором применяется токен.

Константа `KEY` хранит ключ, который будет применяться для создания токена.

И метод `GetSymmetricSecurityKey()` возвращает ключ безопасности, который применяется для генерации токена. Для генерации токена нам необходим объект класса **SecurityKey**. В качестве такого здесь выступает объект производного класса **SymmetricSecurityKey**, в конструктор которого передается массив байт, созданный по секретному ключу.

Чтобы указать, что приложение для аутентификации будет использовать токена, в метод **AddAuthentication()** передается значение константы **JwtBearerDefaults.AuthenticationScheme**.

```
1  builder.Services.AddAuthentication(JwtBearerDefaults.AuthenticationScheme)
```

Конфигурация и валидация токена

С помощью метода **AddJwtBearer()** в приложение добавляется конфигурация токена. Для конфигурации токена применяется объект **JwtBearerOptions**, который позволяет с помощью свойств настроить работу с токеном. Данный объект имеет множество свойств. Здесь же использовано только свойство **TokenValidationParameters**, которое задает параметры валидации токена.

```
builder.Services.AddAuthentication(JwtBearerDefaults.AuthenticationScheme)
    .AddJwtBearer(options =>
    {
        options.TokenValidationParameters = new TokenValidationParameters
        {
            ValidateIssuer = true,
            ValidIssuer = AuthOptions.ISSUER,
            ValidateAudience = true,
            ValidAudience = AuthOptions.AUDIENCE,
            ValidateLifetime = true,
            IssuerSigningKey = AuthOptions.GetSymmetricSecurityKey(),
            ValidateIssuerSigningKey = true,
         };
});
```

Объект **TokenValidationParameters** обладает множеством свойств, которые позволяют настроить различные аспекты валидации токена. В данном случае применяются следующие свойства:

- **ValidateIssuer**: указывает, будет ли валидироваться издатель при валидации токена
- **ValidIssuer**: строка, которая представляет издателя токена
- **ValidateAudience**: указывает, будет ли валидироваться потребитель токена
- **ValidAudience**: строка, которая представляет потребителя токена
- **ValidateLifetime**: указывает, будет ли валидироваться время существования
- **IssuerSigningKey**: представляет ключ безопасности - объект SecurityKey, который будет применяться при генерации токена
- **ValidateIssuerSigningKey**: указывает, будет ли валидироваться ключ безопасности

Здесь устанавливаются наиболее основные свойства. А вообще можно установить кучу других параметров, например, названия claims для ролей и логинов пользователя и т.д.

Генерация токена

Чтобы пользователь мог использовать токен, приложение должно отправить ему этот токен, а перед этим соответственно сгенерировать токен. И для генерации токена здесь предусмотрена типовая конечная точка "/login":

```
app.Map("/login/{username}", (string username) =>
{
    var claims = new List<Claim> {new Claim(ClaimTypes.Name, username) };
    var jwt = new JwtSecurityToken(
            issuer: AuthOptions.ISSUER,
            audience: AuthOptions.AUDIENCE,
            claims: claims,
            expires: DateTime.UtcNow.Add(TimeSpan.FromMinutes(2)), // время действи
            signingCredentials: new SigningCredentials(AuthOptions.GetSymmetricSecu

    return new JwtSecurityTokenHandler().WriteToken(jwt);
});
```

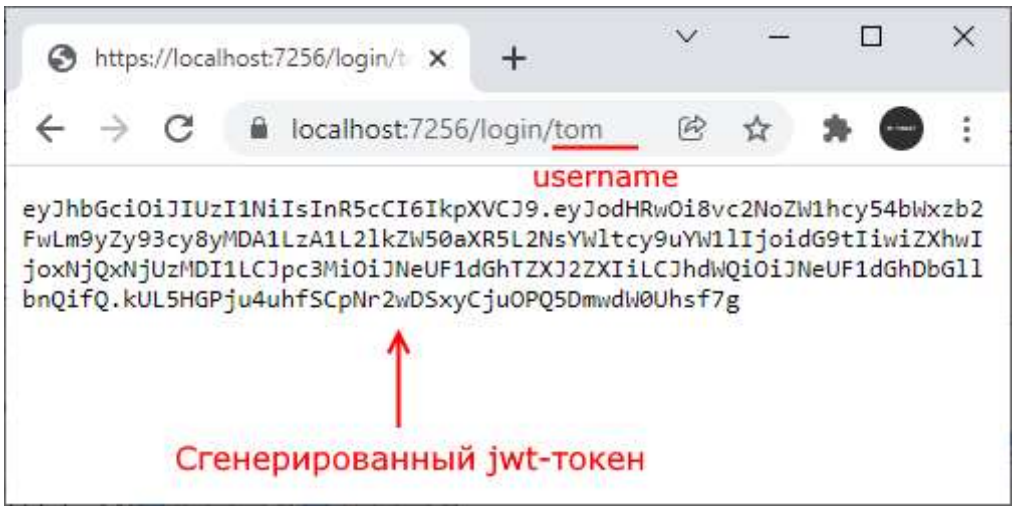
Для простоты конечная точка через параметр маршрута "username" получает некоторый логин пользователя и применяет его для генерации токена. На данном этапе для простоты мы пока ничего не проверяем, валидный ли это логин, что это за логин, пока просто смотрим, как генерировать токен.

Для создания токена применяется конструктор **JwtSecurityToken**. Одним из параметров служит список объектов Claim. Объекты Claim служат для хранения некоторых данных о пользователе, описывают пользователя. Затем эти данные можно применять для аутентификации. В данном случае добавляем в список один Claim, который хранит логин пользователя.

Затем собственно создаем JWT-токен, передавая в конструктор JwtSecurityToken соответствующие параметры. Обратите внимание, что для инициализации токена применяются все те же константы и ключ безопасности, которые определены в классе AuthOptions и которые использовались для конфигурации настроек в методе AddJwtBearer().

В конце посредством метода `JwtSecurityTokenHandler().WriteToken(jwt)` создается сам токен , который отправляется клиенту.

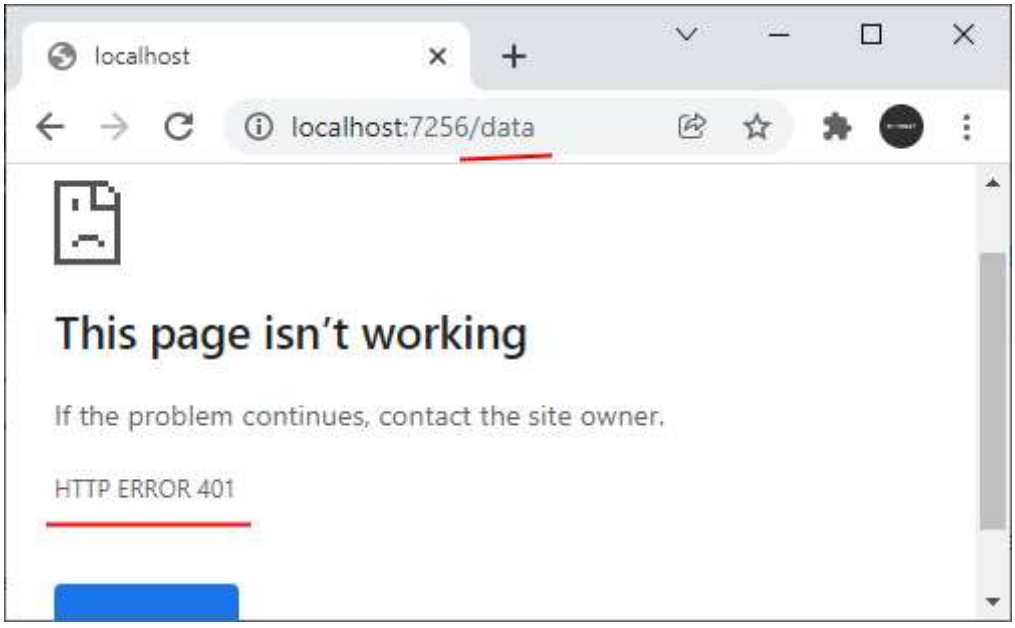
Для тестирования генерации токена обратимся к этой конечной точке:



При обращении к конечной точке "/login" (например, по пути "/login/tom", где "tom" предствляет параметр "username") приложение сгенерирует нам jwt-токен, который нам необходимо отправлять для доступа к ресурсам приложения с защщенным доступом. Например, в коде также определена еще одна конечная точка "/data":

```
1  app.Map("/data", [Authorize] (HttpContext context) => $"Hello World!");
```

Она применяет атрибут **Authorize**, соответственно доступ к ней ограничен только для аутентифицированных пользователей, которые имеют токен. Например, если мы попытаемся обратиться по пути "/data", мы столкнемся с ошибкой 401 (Unauthorized) - доступ не авторизован:



Поэтому для обращения к этому ресурсу (и ко всем другим ресурсам, к которым имеют доступ только аутентифицированные пользователи) необходимо посылать полученный токен в запросе в заголовое **Authorization**:

```
1  "Authorization": "Bearer " + token  // token - полученный ранее jwt-токен
```

В следующей статье рассмотрим, как применять токен для доступа к ресурсам.

Назад **Содержание** **Вперед**

Оля Симоненко • 6 месяцев назад

Всем привет!"

2 • Ответить • Поделиться ›

Gavri Ramlik • 3 месяца назад

Что за класс Claims?

• Ответить • Поделиться ›

Long Landing ➚ Gavri Ramlik • 3 месяца назад

Чуть дальше расскажут

https://metanit.com/sharp/a...

• Ответить • Поделиться ›

Nik Faraday • 7 месяцев назад • edited

Кто такой издатель? Почему это нигде не объясняется? По крайней мере в главе для аутентификации и авторизации

• Ответить • Поделиться ›

phisey ➚ Nik Faraday • 3 месяца назад • edited

Ну допустим это у тебя микросервис. Тогда издатель ISSUER - имя этого микросервиса. А пользоваться (вызывать методы контроллера) могут два других микросервиса. Ты сгенеришь токены для этих двух микросервисов и в полях AUDIENCE укажешь имена этих микросервисов. Но по факту это просто строки, указывающие на того кто генерит токен, и кто должен пользоваться токеном.

2 • Ответить • Поделиться ›

Metanit Модератор ➚ Nik Faraday • 7 месяцев назад

в реальности это может быть произвольная строка

2 • Ответить • Поделиться ›

Перевод на карту
Номер карты:
4048415020898850

Номер карты:
4890494751804113

Вконтакте | Телеграм | Twitter | Канал сайта на youtube | Помощь сайту

Контакты для связи: metanit22@mail.ru