


ASP.NET Core SignalR의 인증 및 권한 부여

2019 년 12 월 5 일 • 읽는 데 9 분 •  +4

이 기사에서

[SignalR 허브에 연결하는 사용자 인증](#)

[사용자에게 허브 및 허브 방법에 액세스 할 수있는 권한 부여](#)

[추가 자료](#)

으로 [앤드류 스탠튼](#) - 간호사

[샘플 코드보기 또는 다운로드 \(다운로드 방법\)](#)

SignalR 허브에 연결하는 사용자 인증

SignalR을 [ASP.NET Core 인증](#) 과 함께 사용 하여 사용자를 각 연결에 연결할 수 있습니다. 허브에서는 `HubConnectionContext.User` 속성 에서 인증 데이터에 액세스 할 수 있습니다. 인증을 통해 허브는 사용자와 관련된 모든 연결에서 메서드를 호출 할 수 있습니다. 자세한 내용 은 [SignalR에서 사용자 및 그룹 관리를 참조 하십시오](#) . 여러 연결이 단일 사용자와 연관 될 수 있습니다.

다음은 `Startup.ConfigureSignalR` 및 ASP.NET Core 인증을 사용 하는 예입니다 .

썬#

부

```
public void Configure(IApplicationBuilder app)
{
    ...

    app.UseStaticFiles();

    app.UseRouting();

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

    app.UseEndpoints(endpoints =>
    {
        endpoints.MapHub<ChatHub>("/chat");
        endpoints.MapControllerRoute("default", "{controller=Home}/{action=Index}/{id?}");
    });
}
```

쿠키 인증

브라우저 기반 앱에서 쿠키 인증을 사용하면 기존 사용자 자격 증명이 자동으로 SignalR 연결로 이동할 수 있습니다. 브라우저 클라이언트를 사용할 때 추가 구성이 필요하지 않습니다. 사용자가 앱에 로그인 한 경우 SignalR 연결은 자동으로 인증을 상속합니다.


쿠키는 액세스 토큰을 보내는 브라우저 별 방법이지만 브라우저가 아닌 클라이언트가 이를 보낼 수 있습니다. 사용하는 경우 [.NET 클라이언트](#)의 `Cookies` 속성은 구성할 수 있습니다. `.WithUrl` 쿠키를 제공하기 위해 호출. 그러나 .NET 클라이언트에서 쿠키 인증을 사용하려면 앱에서 쿠키에 대한 인증 데이터를 교환하기 위한 API를 제공해야 합니다.

Bearer 토큰 인증

클라이언트는 쿠키를 사용하는 대신 액세스 토큰을 제공할 수 있습니다. 서버는 토큰의 유효성을 검사하고 이를 사용하여 사용자를 식별합니다. 이 유효성 검사는 연결이 설정된 경우에만 수행됩니다. 연결이 지속되는 동안 서버는 토큰 해지를 확인하기 위해 자동으로 유효성을 다시 확인하지 않습니다.

서버에서 베어러 토큰 인증은 [JWT 베어러 미들웨어](#)를 사용하여 구성됩니다.

JavaScript 클라이언트에서는 [accessTokenFactory](#) 옵션을 사용하여 토큰을 제공할 수 있습니다.

TypeScript	 부
<pre>// Connect, using the token we got. this.connection = new signalR.HubConnectionBuilder() .withUrl("/hubs/chat", { accessTokenFactory: () => this.loginToken }) .build();</pre>	

.NET 클라이언트에는 토큰을 구성하는 데 사용할 수 있는 유사한 [AccessTokenProvider](#) 속성이 있습니다.

C#	 부
<pre>var connection = new HubConnectionBuilder() .WithUrl("https://example.com/chathub", options => { options.AccessTokenProvider = () => Task.FromResult(_myAccessToken); }) .Build();</pre>	

❗ 노트

The access token function you provide is called before **every** HTTP request made by SignalR. If you need to renew the token in order to keep the connection active (because it may expire during the connection), do so from within this function and return the updated token.

In standard web APIs, bearer tokens are sent in an HTTP header. However, SignalR is unable to set these headers in browsers when using some transports. When using WebSockets and Server-Sent Events, the token is transmitted as a query string parameter. To support this on the server, additional configuration is required:

```
C# Copy

public void ConfigureServices(IServiceCollection services)
{
    services.AddDbContext<ApplicationDbContext>(options =>
options.UseSqlServer(Configuration.GetConnectionString("DefaultConnection"))
);

    services.AddIdentity<ApplicationUser, IdentityRole>()
        .AddEntityFrameworkStores<ApplicationDbContext>()
        .AddDefaultTokenProviders();

    services.AddAuthentication(options =>
    {
        // Identity made Cookie authentication the default.
        // However, we want JWT Bearer Auth to be the default.
        options.DefaultAuthenticateScheme =
JwtBearerDefaults.AuthenticationScheme;
        options.DefaultChallengeScheme =
JwtBearerDefaults.AuthenticationScheme;
    })
        .AddJwtBearer(options =>
        {
            // Configure the Authority to the expected value for your
authentication provider
            // This ensures the token is appropriately validated
            options.Authority = /* TODO: Insert Authority URL here */;

            // We have to hook the OnMessageReceived event in order to
            // allow the JWT authentication handler to read the access
            // token from the query string when a WebSocket or
            // Server-Sent Events request comes in.

            // Sending the access token in the query string is required due
to
            // a limitation in Browser APIs. We restrict it to only calls to
the
            // SignalR hub in this code.
            // See
https://docs.microsoft.com/aspnet/core/signalr/security#access-token-logging

```

```
// for more information about security considerations when using
// the query string to transmit the access token.
options.Events = new JwtBearerEvents
{
    OnMessageReceived = context =>
    {
        var accessToken = context.Request.Query["access_token"];

        // If the request is for our hub...
        var path = context.HttpContext.Request.Path;
        if (!string.IsNullOrEmpty(accessToken) &&
            (path.StartsWithSegments("/hubs/chat")))
        {
            // Read the token out of the query string
            context.Token = accessToken;
        }
        return Task.CompletedTask;
    }
};
});
```

```
services.AddMvc().SetCompatibilityVersion(CompatibilityVersion.Version_2_1);
```

```
services.AddSignalR();
```

```
// Change to use Name as the user identifier for SignalR
// WARNING: This requires that the source of your JWT token
// ensures that the Name claim is unique!
// If the Name claim isn't unique, users could receive messages
// intended for a different user!
services.AddSingleton<IUserIdProvider, NameUserIdProvider>();
```

```
// Change to use email as the user identifier for SignalR
// services.AddSingleton<IUserIdProvider, EmailBasedUserIdProvider>();
```

```
// WARNING: use *either* the NameUserIdProvider *or* the
// EmailBasedUserIdProvider, but do not use both.
```

```
}
```

If you would like to see code comments translated to languages other than English, let us know in [this GitHub discussion issue](#).

❗ Note

The query string is used on browsers when connecting with WebSockets and Server-Sent Events due to browser API limitations. When using HTTPS, query string values are secured by the TLS connection. However, many servers log query string values. For more information, see [Security considerations in ASP.NET Core](#)

SignalR. SignalR uses headers to transmit tokens in environments which support them (such as the .NET and Java clients).

Cookies vs. bearer tokens

Cookies are specific to browsers. Sending them from other kinds of clients adds complexity compared to sending bearer tokens. Consequently, cookie authentication isn't recommended unless the app only needs to authenticate users from the browser client. Bearer token authentication is the recommended approach when using clients other than the browser client.

Windows authentication

If [Windows authentication](#) is configured in your app, SignalR can use that identity to secure hubs. However, to send messages to individual users, you need to add a custom User ID provider. The Windows authentication system doesn't provide the "Name Identifier" claim. SignalR uses the claim to determine the user name.

Add a new class that implements `IUserIdProvider` and retrieve one of the claims from the user to use as the identifier. For example, to use the "Name" claim (which is the Windows username in the form `[Domain]\[Username]`), create the following class:

C#

 Copy

```
public class NameUserIdProvider : IUserIdProvider
{
    public string GetUserId(HubConnectionContext connection)
    {
        return connection.User?.Identity?.Name;
    }
}
```

Rather than `ClaimTypes.Name`, you can use any value from the user (such as the Windows SID identifier, and so on).

ⓘ Note

The value you choose must be unique among all the users in your system. Otherwise, a message intended for one user could end up going to a different user.

Register this component in your `Startup.ConfigureServices` method.

C#

 Copy

```
public void ConfigureServices(IServiceCollection services)
{
    // ... other services ...

    services.AddSignalR();
    services.AddSingleton<IUserIdProvider, NameUserIdProvider>();
}
```

In the .NET Client, Windows Authentication must be enabled by setting the [UseDefaultCredentials](#) property:

C#

 Copy

```
var connection = new HubConnectionBuilder()
    .WithUrl("https://example.com/chathub", options =>
    {
        options.UseDefaultCredentials = true;
    })
    .Build();
```

Windows authentication is supported in Internet Explorer and Microsoft Edge, but not in all browsers. For example, in Chrome and Safari, attempting to use Windows authentication and WebSockets fails. When Windows authentication fails, the client attempts to fall back to other transports which might work.

Use claims to customize identity handling

An app that authenticates users can derive SignalR user IDs from user claims. To specify how SignalR creates user IDs, implement `IUserIdProvider` and register the implementation.

The sample code demonstrates how you would use claims to select the user's email address as the identifying property.

Note

The value you choose must be unique among all the users in your system. Otherwise, a message intended for one user could end up going to a different user.

C#

 Copy

```
public class EmailBasedUserIdProvider : IUserIdProvider
{
    public virtual string GetUserId(HubConnectionContext connection)
    {
        return connection.User?.FindFirst(ClaimTypes.Email)?.Value;
    }
}
```

The account registration adds a claim with type `ClaimsTypes.Email` to the ASP.NET identity database.

C#

 Copy

```
// create a new user
var user = new ApplicationUser { UserName = Input.Email, Email = Input.Email };
var result = await _userManager.CreateAsync(user, Input.Password);

// add the email claim and value for this user
await _userManager.AddClaimAsync(user, new Claim(ClaimTypes.Email, Input.Email));
```

Register this component in your `Startup.ConfigureServices`.

C#

 Copy

```
services.AddSingleton<IUserIdProvider, EmailBasedUserIdProvider>();
```

Authorize users to access hubs and hub methods

By default, all methods in a hub can be called by an unauthenticated user. To require authentication, apply the [Authorize](#) attribute to the hub:

C#

 Copy

```
[Authorize]
public class ChatHub: Hub
{
}
```

You can use the constructor arguments and properties of the `[Authorize]` attribute to restrict access to only users matching specific [authorization policies](#). For example, if you have a custom authorization policy called `MyAuthorizationPolicy` you can ensure that only users matching that policy can access the hub using the following code:

C#

 Copy

```
[Authorize("MyAuthorizationPolicy")]
public class ChatHub : Hub
{
}
```

Individual hub methods can have the `[Authorize]` attribute applied as well. If the current user doesn't match the policy applied to the method, an error is returned to the caller:

C#

 Copy

```
[Authorize]
public class ChatHub : Hub
{
    public async Task Send(string message)
    {
        // ... send a message to all users ...
    }

    [Authorize("Administrators")]
    public void BanUser(string userName)
    {
        // ... ban a user from the chat room (something only Administrators
        // can do) ...
    }
}
```

Use authorization handlers to customize hub method authorization

SignalR provides a custom resource to authorization handlers when a hub method requires authorization. The resource is an instance of `HubInvocationContext`. The `HubInvocationContext` includes the `HubCallerContext`, the name of the hub method being invoked, and the arguments to the hub method.

Consider the example of a chat room allowing multiple organization sign-in via Azure Active Directory. Anyone with a Microsoft account can sign in to chat, but only members of the owning organization should be able to ban users or view users' chat histories. Furthermore, we might want to restrict certain functionality from certain users. Using the updated features in ASP.NET Core 3.0, this is entirely possible. Note how the `DomainRestrictedRequirement` serves as a custom `IAuthorizationRequirement`. Now that the `HubInvocationContext` resource parameter is being passed in, the internal logic can

inspect the context in which the Hub is being called and make decisions on allowing the user to execute individual Hub methods.

C#

 Copy

```
[Authorize]
public class ChatHub : Hub
{
    public void SendMessage(string message)
    {
    }

    [Authorize("DomainRestricted")]
    public void BanUser(string username)
    {
    }

    [Authorize("DomainRestricted")]
    public void ViewUserHistory(string username)
    {
    }
}

public class DomainRestrictedRequirement :
    AuthorizationHandler<DomainRestrictedRequirement, HubInvocationContext>,
    IAuthorizationRequirement
{
    protected override Task
    HandleRequirementAsync(AuthorizationHandlerContext context,
        DomainRestrictedRequirement requirement,
        HubInvocationContext resource)
    {
        if (IsUserAllowedToDoThis(resource.HubMethodName,
            context.User.Identity.Name) &&
            context.User.Identity.Name.EndsWith("@microsoft.com"))
        {
            context.Succeed(requirement);
        }
        return Task.CompletedTask;
    }

    private bool IsUserAllowedToDoThis(string hubMethodName,
        string currentUsername)
    {
        return !(currentUsername.Equals("asdf42@microsoft.com") &&
            hubMethodName.Equals("banUser",
                StringComparison.OrdinalIgnoreCase));
    }
}
```

In `Startup.ConfigureServices`, add the new policy, providing the custom `DomainRestrictedRequirement` requirement as a parameter to create the

DomainRestricted policy.

C#

 Copy

```
public void ConfigureServices(IServiceCollection services)
{
    // ... other services ...

    services
        .AddAuthorization(options =>
        {
            options.AddPolicy("DomainRestricted", policy =>
            {
                policy.Requirements.Add(new DomainRestrictedRequirement());
            });
        });
}
```

앞의 예에서 `DomainRestrictedRequirement` 클래스는 해당 요구 사항에 대해 하나 `IAuthorizationRequirement` 이며 고유 `AuthorizationHandler` 합니다. 이 두 구성 요소를 별도의 클래스로 분할하여 우려 사항을 분리 할 수 있습니다. 예제의 접근 방식의 이점은 `AuthorizationHandler` 요구 사항과 처리기가 동일하기 때문에 시작 중에 를 주입 할 필요가 없다는 것입니다.

추가 자료

- [ASP.NET Core의 Bearer 토큰 인증](#)
- [리소스 기반 인증](#)

Is this page helpful?

 Yes  No