

HTTP-cookie – это небольшой фрагмент данных (пара «ключ=значение»), который сервер отправляет браузеру в заголовке ответа `Set-Cookie`. Браузер сохраняет этот фрагмент и автоматически шлёт его обратно на тот же домен при последующих запросах в заголовке `Cookie`.

Зачем используют

1. **Сессии и авторизация.** После успешного логина сервер ставит в cookie идентификатор сессии, по которому узнаёт пользователя в следующих запросах.
2. **Персонализация.** Хранение настроек (тема, язык), корзины покупок, фильтров.
3. **Отслеживание.** Счётчики посещений, аналитика (третьесторонние cookie).

Основные атрибуты cookie

Атрибут	Назначение
Expires	Дата и время истечения срока жизни cookie. После этого браузер её удалит.
Max-Age	Альтернатива Expires : время жизни в секундах после установки.
HttpOnly	Запрет доступа к cookie из JavaScript (<code>document.cookie</code>). Защита от XSS.
Secure	Отправка только по HTTPS-соединению.
SameSite	Контроль кросс-сайтовых запросов:

- Strict – cookie не отправляются при переходе по ссылке с другого сайта.
- Lax – отправляются при «нулевых» навигациях (GET-переходы).
- None – отправляются всегда (требует Secure).

Простой пример в ASP.NET Core Minimal API

Ниже – мини-пример, как в Minimal API установить, прочитать и удалить cookie.

csharp

КопироватьРедактировать

```
var builder = WebApplication.CreateBuilder(args);
var app = builder.Build();

// 1. Установка cookie
app.MapGet("/cookie/set", (HttpResponse resp) =>
{
    resp.Cookies.Append(
        "MyCookie",           // имя
        "HelloWorld",         // значение
        new CookieOptions
        {
            HttpOnly = true,    // недоступно из JS
            Secure = false,     // локально можно false
            Expires = DateTimeOffset.UtcNow.AddMinutes(5),
            SameSite = SameSiteMode.Lax
        });
    return Results.Ok("Cookie установлено");
});

// 2. Чтение cookie
app.MapGet("/cookie/read", (HttpRequest req) =>
{
    if (req.Cookies.TryGetValue("MyCookie", out var val))
        return Results.Ok($"MyCookie = {val}");
    return Results.NotFound("Cookie не найдена");
});

// 3. Удаление cookie
app.MapGet("/cookie/delete", (HttpResponse resp) =>
{
    resp.Cookies.Delete("MyCookie");
    return Results.Ok("Cookie удалено");
});

app.Run();
```

Как это работает «под капотом»:

1. Клиент вызывает `GET /cookie/set`.
2. Сервер отвечает с заголовком:

Set-Cookie: MyCookie=HelloWorld; Expires=...; HttpOnly; SameSite=Lax

3. Браузер сохраняет cookie и при следующих запросах к этому же хосту добавляет:

Cookie: MyCookie=HelloWorld

4. В Minimal API мы читаем её из `req.Cookies`.

С чем «едят» cookie

1. **Авторизация:** после логина ставим cookie `session-id`, по нему узнаём пользователя.
2. **Настройки:** тема (`theme=dark`), локализация (`lang=ru`).
3. **Корзина:** временное хранение `id` товаров до оформления заказа.
4. **CSRF-защита:** ASP.NET Core генерирует анти-фожери-токен в виде cookie + скрытого поля формы.

Итог:

- Cookie – это просто способ хранить небольшие данные на стороне клиента и автоматически передавать их серверу.
- В ASP.NET Core вы можете легко управлять ими через `Request.Cookies` и `Response.Cookies`.
- Обязательно настраивайте `HttpOnly`, `Secure` и `SameSite` согласно потребностям безопасности.

Пример

Чтобы **использовать** куки в приложении, обычно делают две вещи:

1. **Хранение пользовательского сеанса (session)** – чтобы после логина не запрашивать пароль на каждом запросе.
2. **Сохранение пользовательских настроек** (тема, язык и т. п.) – чтобы при следующем заходе сразу отображать, скажем, «тёмную» тему.

Ниже – пример минимального приложения на ASP.NET Core 7 (Minimal API), где мы:

- Конфигурируем **Cookie-аутентификацию** (без Identity).
- Реализуем эндпоинты `/login`, `/logout` и защищённый `/secret`.
- При логине в куки сохраняем `claim` с именем пользователя.
- Покажем, как читать из куки текущего пользователя и как ограничить доступ.

```
using Microsoft.AspNetCore.Authentication.Cookies;
using Microsoft.AspNetCore.Authorization;
using System.Security.Claims;

var builder = WebApplication.CreateBuilder(args);

// 1) Добавляем схему куки-аутентификации
builder.Services
    .AddAuthentication(CookieAuthenticationDefaults.AuthenticationScheme)
    .AddCookie(opts =>
    {
        opts.LoginPath = "/login";    // при неавторизованном запросе – редирект сюда
        opts.LogoutPath = "/logout";  // по выходу
        opts.ExpireTimeSpan = TimeSpan.FromMinutes(30);
        opts.SlidingExpiration = true;
        opts.Cookie.HttpOnly = true;
        opts.Cookie.SecurePolicy = CookieSecurePolicy.None; // в проде – Always
        opts.Cookie.SameSite = SameSiteMode.Lax;
    });

builder.Services.AddAuthorization();

var app = builder.Build();

// 2) Включаем middleware аутентификации/авторизации
app.UseAuthentication();
app.UseAuthorization();

// 3) Эндпоинт логина: проверяем креды и создаём куки
app.MapPost("/login", async (HttpContext ctx, LoginDto dto) =>
{
    // В реальности – проверка в БД. Здесь просто заглушка:
    if (dto.Username == "admin" && dto.Password == "123")
    {
        var claims = new[]
        {
            new Claim(ClaimTypes.Name, dto.Username),
            new Claim(ClaimTypes.Role, "Administrator")
        };
        var principal = new ClaimsPrincipal(
            new ClaimsIdentity(claims, CookieAuthenticationDefaults.AuthenticationScheme)
        );

        await ctx.SignInAsync(principal);
        return Results.Ok("Вошли как admin");
    }

    return Results.Unauthorized();
});

// 4) Эндпоинт выхода: удаляет куки
app.MapPost("/logout", async (HttpContext ctx) =>
{
    await ctx.SignOutAsync();
    return Results.Ok("Вы вышли");
});

// 5) Защищённый эндпоинт – только для аутентифицированных
app.MapGet("/secret", [Authorize] (ClaimsPrincipal user) =>
{
    // user.Identity.Name берётся из куки
    return Results.Ok($"Привет, {user.Identity.Name}! Это секретная страница.");
});
```

```
});  
  
// 6) DTO для логина  
public record LoginDto(string Username, string Password);  
  
app.Run();
```

HTTP

https://localhost:7256/login

Save

Share

</>

POST

https://localhost:7256/login

Send

↺

Params

Authorization

Headers (10)

Body

Scripts

Settings

Cookies

☐ none

☐ form-data

☐ x-www-form-urlencoded

☒ raw

☐ binary

☐ GraphQL

JSON

Beautify

```
1 {
2   "Username" : "admin",
3   "Password" : "123"
4 }
```

Body

Cookies (1)

Headers (8)

Test Results

200 OK

114 ms

742 B

🌐

⋮

{ } JSON

Preview

Visualize

↺

≡

🔍

📄

🔗

```
1 "Вошли как admin"
```

Postbot

Runner

Start Proxy

Cookies

Vault

Trash

⌵

?

POST https://localhost:7256/

GET https://localhost:7256/secret

+

No environment

🔗

HTTP

https://localhost:7256/secret

Save

Share

</>

GET

https://localhost:7256/secret

Send

↺

Params

Authorization

Headers (8)

Body

Scripts

Settings

Cookies

Query Params

	Key	Value	Description	⋮	Bulk Edit
	Key	Value	Description		

Body

Cookies (1)

Headers (4)

Test Results

200 OK

13 ms

214 B

🌐

⋮

{ } JSON

Preview

Visualize

↺

≡

🔍

📄

🔗

```
1 "Привет, admin! Это секретная страница."
```

Postbot

Runner

Start Proxy

Cookies

Vault

Trash

⌵

?

Как это работает

1. Конфигурация

- В `AddAuthentication().AddCookie()` задаёте, как и куда ставить куки, срок жизни, защитные флаги.
- `UseAuthentication()` и `UseAuthorization()` подключают middleware, которые будут перехватывать запросы.

2. Логин

- Клиент делает `POST /login` с JSON:

```
{ "username": "admin", "password": "123" }
```

Сервер проверяет данные, создаёт `ClaimsPrincipal` (набор `claims`, например имя и роль) и вызывает

```
await ctx.SignInAsync(principal);
```

Это добавляет в ответ заголовок

```
Set-Cookie: .AspNetCore.Cookies=<токен>;  
HttpOnly; SameSite=Lax; Expires=...
```

- **Защищённый доступ**
 - Любой `GET /secret` попадёт в middleware аутентификации:
 - Если в запросе есть валидный куки, он создаёт `HttpContext.User` из `claims`.
 - Если нет — отдаст 401 или перенаправит на `/login` (в зависимости от настроек).
- **Чтение данных**
 - Внутри защищённого эндпоинта вы получаете `ClaimsPrincipal user`.
 - `user.Identity.Name` — имя пользователя из `claim`, другие `claims` можно вытаскивать через `user.FindFirst(...)`.
- **Logout**
 - `SignOutAsync()` очищает куки — браузер больше не будет слать его, и пользователь «разлогинится».

Пример с добавлением данных: темы и языка

```
using Microsoft.AspNetCore.Authentication;  
using Microsoft.AspNetCore.Authentication.Cookies;  
using Microsoft.AspNetCore.Authorization;  
using System.Security.Claims;  
  
var builder = WebApplication.CreateBuilder(args);  
  
builder.Services.AddOpenApi();  
builder.Services.AddAuthentication(CookieAuthenticationDefaults.AuthenticationScheme)  
    .AddCookie(opts =>  
    {  
        opts.LoginPath = "/login";  
        opts.LogoutPath = "/logout";  
        opts.ExpireTimeSpan = TimeSpan.FromMinutes(30);  
        opts.SlidingExpiration = true;  
        opts.Cookie.HttpOnly = true;  
        opts.Cookie.SecurePolicy = CookieSecurePolicy.None; // в проде — Always  
        opts.Cookie.SameSite = SameSiteMode.Lax;  
    });  
  
builder.Services.AddAuthorization();  
  
var app = builder.Build();  
app.UseAuthentication();  
app.UseAuthorization();  
  
if (app.Environment.IsDevelopment())  
{  
    app.MapOpenApi();  
}  
  
app.UseHttpsRedirection();  
  
app.MapPost("/login", async (HttpContext ctx, LoginDto dto) =>  
{  
    // В реальности — проверка в БД. Здесь просто заглушка:  
    if (dto.Username == "admin" && dto.Password == "123")  
    {  
        var claims = new[]  
        {  
            new Claim(ClaimTypes.Name, dto.Username),  
            new Claim(ClaimTypes.Role, "Administrator"),  
            // добавляем настройки  
            new Claim("theme", "dark"), // или dto.Theme  
            new Claim("lang", "ru-RU") // или dto.Lang  
        };  
        var principal = new ClaimsPrincipal(  
            new ClaimsIdentity(claims, CookieAuthenticationDefaults.AuthenticationScheme)  
        );
```

```
        await ctx.SignInAsync(principal);
        return Results.Ok("Вошли как admin");
    }

    return Results.Unauthorized();
});
app.MapPost("/logout", async (HttpContext ctx) =>
{
    await ctx.SignOutAsync();
    return Results.Ok("Вы вышли");
});

app.MapGet("/secret", [Authorize] (ClaimsPrincipal user) =>
{
    return Results.Ok($"Привет, {user.Identity.Name}! Это секретная страница.");
});

app.MapGet("/profile", [Authorize] (ClaimsPrincipal user) =>
{
    var theme = user.FindFirst("theme")?.Value ?? "default";
    var lang = user.FindFirst("lang")?.Value ?? "en-US";
    return Results.Ok(new
    {
        user = user.Identity.Name,
        theme,
        lang
    });
});

app.Run();

public record LoginDto(string Username, string Password);
```