

# Современные платформы прикладной разработки

---

АУТЕНТИФИКАЦИЯ И АВТОРИЗАЦИЯ



# Общая информация

---

АУТЕНТИФИКАЦИЯ И АВТОРИЗАЦИЯ

# Аутентификация и авторизация

---

***Аутентификация*** - процедура проверки подлинности, например: проверка подлинности пользователя путём сравнения введённого им пароля с паролем в базе данных пользователей

# Аутентификация и авторизация

---

**Авторизация** - предоставление определённому лицу или группе лиц прав на выполнение определённых действий, а также процесс проверки (подтверждения) данных прав при попытке выполнения этих действий.

# Аутентификация и авторизация

---

ASP.NET Core имеет встроенную поддержку аутентификации на основе **куки**.

# Аутентификация и авторизация

---

В ASP.NET Core аутентификация обрабатывается службой ***IAuthenticationService***, которая регистрируется в качестве сервиса и используется промежуточным программным обеспечением (Middleware) аутентификации.

# Аутентификация и авторизация

---

Компонент `middleware` аутентификации, сериализует данные пользователя в зашифрованные аутентификационные куки и передает их на сторону клиента.

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

# Аутентификация и авторизация

---

Свойство **User** является объектом класса ***ClaimsPrincipal***.

***ClaimsPrincipal*** позволяет работать с объектами ***claim*** (**Утверждения**).



# Аутентификация и авторизация

---

***Claim*** - это информация о пользователе, которая используется в контексте операций аутентификации и авторизации.

# Некоторые свойства объекта Claim

---

**Type** представляет собой строку (обычно URI) - семантическое описание того, какого рода информация содержится в claim.

**Subject** - это объект ClaimsIdentity, представляющий пользователя.

**Value** содержит значение claim.

# Методы ClaimsPrincipal

---

- **Identity**: возвращает объект ClaimsIdentity, который реализует интерфейс IIdentity и представляет текущего пользователя
- **FindAll(type) / FindAll(predicate)**: возвращает все объекты claim, которые соответствуют определенному типу или условию
- **FindFirst(type) / FindFirst(predicate)**: возвращает первый объект claim, который соответствует определенному типу или условию
- **HasClaim(type, value) / HasClaim(predicate)**: возвращает значение true, если пользователь имеет claim определенного типа с определенным значением
- **IsInRole(name)**: возвращает значение true, если пользователь принадлежит роли с названием name

# Свойства объекта ClaimsIdentity

- **Claims:** свойство, которое возвращает набор ассоциированных с пользователем объектов claim
- **AddClaim(claim):** добавляет для пользователя объект claim
- **AddClaims(claims):** добавляет набор объектов claim
- **FindAll(predicate):** возвращает все объекты claim, которые соответствуют определенному условию
- **HasClaim(predicate):** возвращает значение true, если пользователь имеет claim, соответствующий определенному условию
- **RemoveClaim(claim):** удаляет объект claim

# Добавление сервиса аутентификации

```
builder.Services.AddAuthentication(JwtBearerDefaults.AuthenticationScheme)
    .AddJwtBearer(JwtBearerDefaults.AuthenticationScheme,
        options => builder.Configuration.Bind("JwtSettings", options))
    .AddCookie(CookieAuthenticationDefaults.AuthenticationScheme,
        options => builder.Configuration.Bind("CookieSettings", options));
```

Код регистрирует службы аутентификации и обработчики для схем аутентификации на основе файлов cookie и JWT.

Параметр метода `AddAuthentication JwtBearerDefaults.AuthenticationScheme` — это имя схемы, используемой по умолчанию, когда конкретная схема не запрашивается.

```
[Authorize]
public IActionResult Claims()
{
    return View(User?.Claims);
}
```

```

@model IEnumerable<System.Security.Claims.Claim>
@{
    ViewData["Title"] = "CLAIMS";
}

<h2 class="bg-primary m-1 p-1 text-white">Claims</h2>

<table class="table table-sm table-bordered">
    <tr>
        <th>Subject</th>
        <th>Issuer</th>
        <th>Type</th>
        <th>Value</th>
    </tr>

    @foreach (var claim in Model.OrderBy(x => x.Type))
    {
        <tr>
            <td>@claim.Subject.Name</td>
            <td>@claim.Issuer</td>
            <td>@claim.Type</td>
            <td>@claim.Value</td>
        </tr>
    }
</table>

```

## Claims

Subject	Issuer	Type	Value
user1@mail.ru	LOCAL AUTHORITY	AspNet.Identity.SecurityStamp	VBYPYAYBY3CXIELYMUCAGQEEYXCUTPY5
user1@mail.ru	LOCAL AUTHORITY	http://schemas.xmlsoap.org/ws/2005/05/identity/claims/emailaddress	user1@mail.ru
user1@mail.ru	LOCAL AUTHORITY	http://schemas.xmlsoap.org/ws/2005/05/identity/claims/name	user1@mail.ru
user1@mail.ru	LOCAL AUTHORITY	http://schemas.xmlsoap.org/ws/2005/05/identity/claims/nameidentifier	66b28a99-a8ca-4477-b18a-a79a406ac116



# Чтение Claim

---

```
var id = User.FindFirst(ClaimTypes.NameIdentifier);
```

# ASP.Net Core Identity

---

АУТЕНТИФИКАЦИЯ И АВТОРИЗАЦИЯ

# ASP.Net Core Identity

---

***ASP.NET Core Identity*** – это система членства, позволяющая регистрировать учетные записи пользователей, регистрировать роли и назначать роли пользователям для реализации механизма аутентификации и авторизации.

# ASP.Net Core Identity

---

Базовый набор интерфейсов и классов, используемых в системе аутентификации находятся в пространстве имен ***Microsoft.AspNetCore.Identity***

## Некоторые классы Microsoft.AspNetCore.Identity

---

***IdentityUser*** – описывает пользователя;

***IdentityRole*** – описывает роль;

***IdentityUserClaim*** – описывает Claim пользователя.

# ASP.Net Core Identity

---

Для работы с пользователями и ролями в пространстве имен `Microsoft.AspNetCore.Identity` описаны **классы менеджеров**.

# ASP.Net Core Identity

---

***UserManager<TUser>*** – управляет пользователями (добавление, удаление, поиск, назначение роли и т.д.);

***RoleManager<TRole>*** - управляет ролями пользователей (добавление, удаление, поиск, роли и т.д.);

***SignInManager<TUser>*** – реализует функции входа в/выхода из системы пользователя и формирует куки аутентификации

# ASP.Net Core Identity

---

Добавление пользователя :

```
var result = await _userManager  
                .CreateAsync(newUser, password)
```

Поиск пользователя по Email

```
var user = await  
            _userManager.FindByEmailAsync("xxx")
```



# ASP.Net Core Identity

---

## Добавление утверждения (Claim)

```
var newClaim = new Claim("company", "Microsoft");  
var result = await _userManager.AddClaimAsync(newUser,newClaim);
```

Сначала создается клеймо в виде пары «имя-значение». Затем клеймо добавляется к пользователю.

# Создание куки аутентификации

```
await _signInManager.SignInAsync(newUser, isPersistent: false);
```

**или**

```
var result = await _signInManager
    .PasswordSignInAsync("name", "password",
        isPersistent: false,
        lockoutOnFailure: false);
```

# Создание куки аутентификации

---

- ❑ Куки создаются при регистрации пользователя в систему.
- ❑ Параметр `isPersistent`, установленный в `false`, означает, что куки аутентификации не сохраняются после того, как пользователь закрыл браузер.
- ❑ Переменная `result` получит результат операции, например, успешное завершение или наоборот, отсутствие пользователя с введенными данными

Microsoft.AspNetCore.Identity.EntityFrameworkCore

---

АУТЕНТИФИКАЦИЯ И АВТОРИЗАЦИЯ

# Identity.EntityFrameworkCore

---

Менеджеры взаимодействуют с хранилищем данных через интерфейсы ***IUserStore<TUser>***, ***IRoleStore<TRole>***

Это значит, что можно использовать любую базу данных, описав соответствующие классы хранилищ.

# Identity.EntityFrameworkCore

---

В пространстве имен

**Microsoft.AspNetCore.Identity.EntityFrameworkCore**

находится готовая реализация хранилищ на базе Entity Framework Core

Здесь описаны классы

☐ **UserStore,**

☐ **RoleStore,**

☐ **IdentityDbContext.**

# Identity.EntityFrameworkCore

---

```
services.AddDefaultIdentity<IdentityUser>(options =>  
    options.SignIn.RequireConfirmedAccount = true)  
    .AddEntityFrameworkStores<ApplicationDbContext>();
```

```
. . .
```

```
app.UseAuthentication();
```

# AddDefaultIdentity

---

Вызов AddDefaultIdentity аналогичен вызову следующего:

- ☐ AddIdentity
- ☐ AddDefaultUI
- ☐ AddDefaultTokenProviders

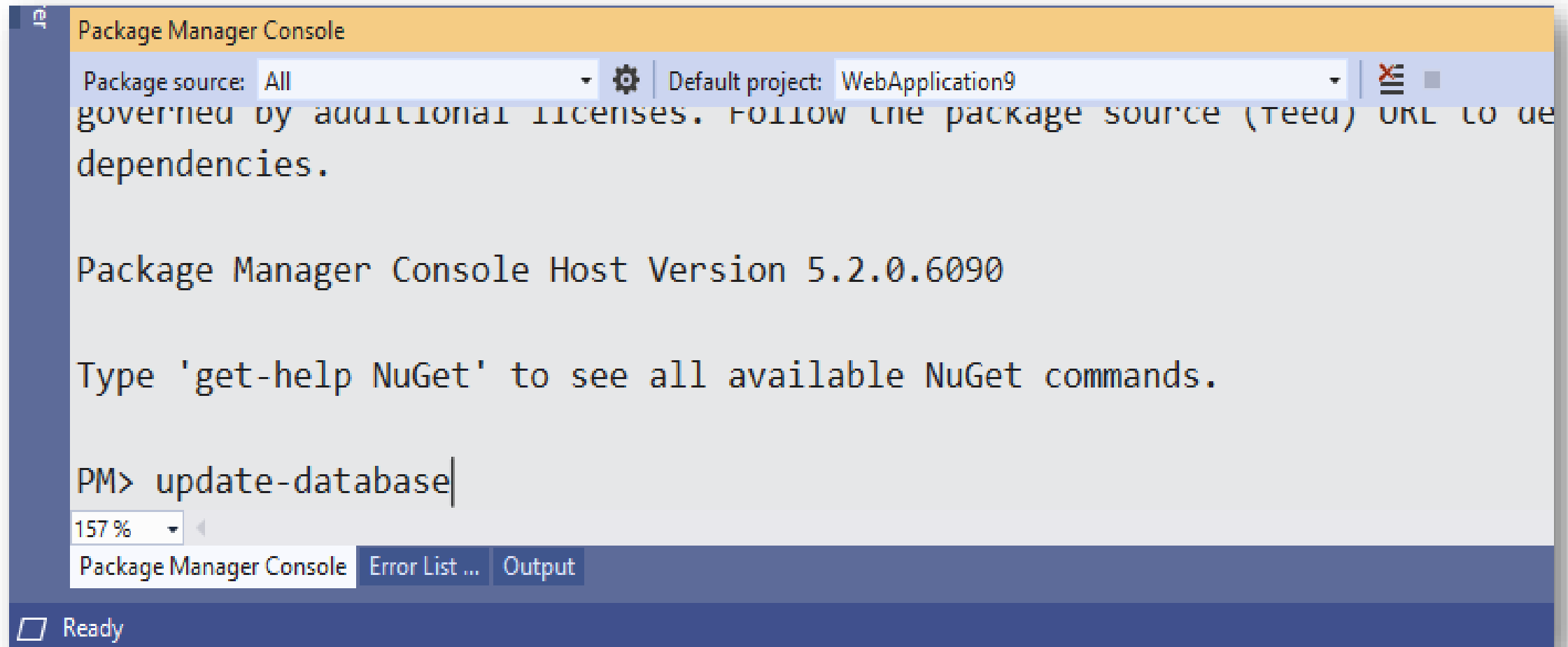


Для добавления сервисов управления ролями можно воспользоваться методом

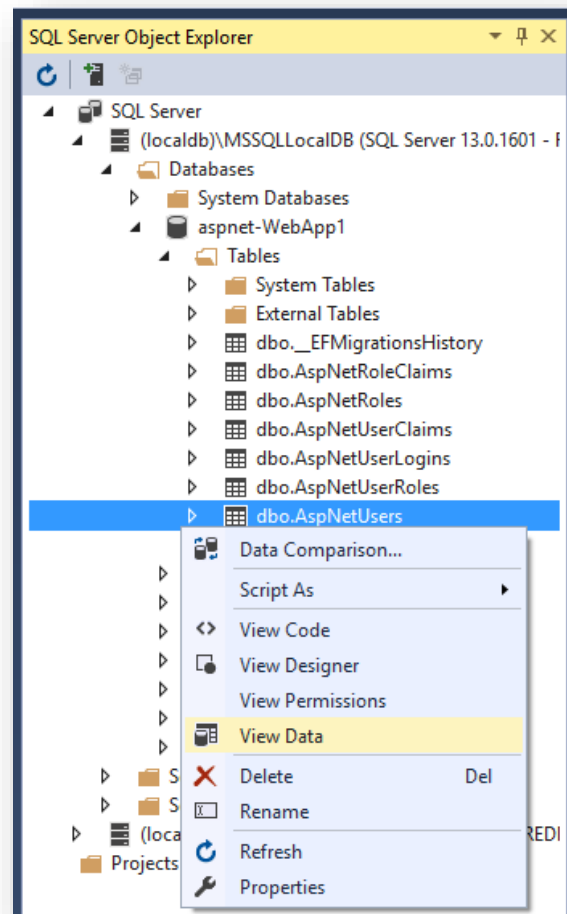
***AddIdentity<TUser,TRole>(IServiceCollection,  
Action<IdentityOptions>)***

Или

```
builder.Services.AddDefaultIdentity<IdentityUser>(options => {  
    options.SignIn.RequireConfirmedAccount = false;})  
    .AddRoles<IdentityRole>()  
    .AddEntityFrameworkStores<ApplicationDbContext>();
```



# Таблицы базы данных

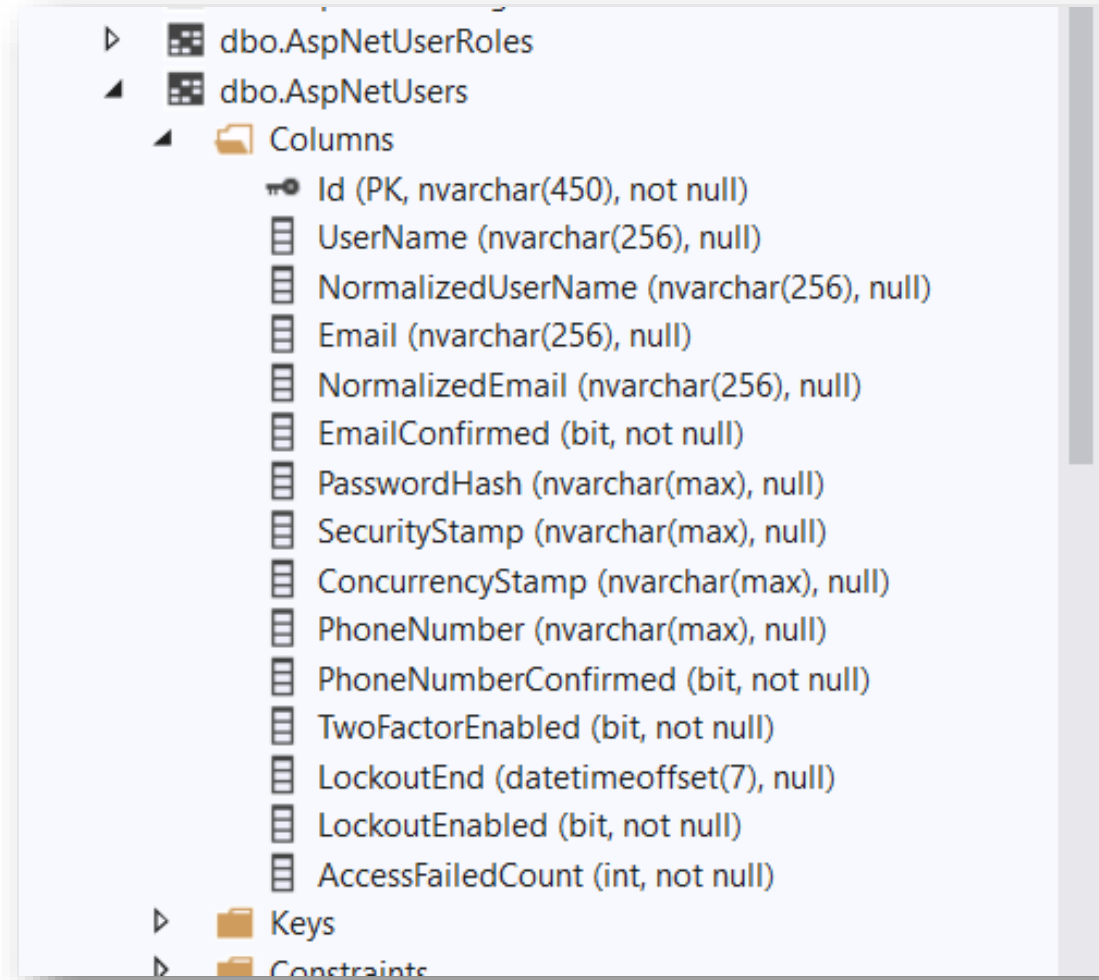


# Пример настройки ограничений пароля

```
services.AddIdentity<IdentityUser, IdentityRole>(opt=> {  
    opt.Password.RequiredLength = 4;  
    opt.Password.RequireLowercase = false;  
    opt.Password.RequireNonAlphanumeric = false;  
    opt.Password.RequireUppercase = false;  
})
```



# ТаблицаAspNetUsers



The screenshot shows a tree view of a database schema. The 'dbo.AspNetUsers' table is expanded, showing its columns and keys. The columns are listed with their data types and nullability. The 'Id' column is marked as the primary key. The 'Keys' and 'Constraints' folders are also visible at the bottom of the tree.

▶	dbo.AspNetUserRoles
▶	dbo.AspNetUsers
▶	Columns
	Id (PK, nvarchar(450), not null)
	UserName (nvarchar(256), null)
	NormalizedUserName (nvarchar(256), null)
	Email (nvarchar(256), null)
	NormalizedEmail (nvarchar(256), null)
	EmailConfirmed (bit, not null)
	PasswordHash (nvarchar(max), null)
	SecurityStamp (nvarchar(max), null)
	ConcurrencyStamp (nvarchar(max), null)
	PhoneNumber (nvarchar(max), null)
	PhoneNumberConfirmed (bit, not null)
	TwoFactorEnabled (bit, not null)
	LockoutEnd (datetimeoffset(7), null)
	LockoutEnabled (bit, not null)
	AccessFailedCount (int, not null)
▶	Keys
▶	Constraints

# Identity.EntityFrameworkCore

---

В приложении ASP.NET MVC обычно не работают напрямую с классом `IdentityDbContext`.

В `IdentityDbContext` описаны только классы, необходимые для системы Identity.

В реальном приложении вы скорее всего будете работать также с другими данными, которые отображаются на предметную область приложения. Поэтому создается свой класс, в нашем случае это `ApplicationDbContext`, который наследуют от класса `IdentityDbContext`.

# Identity.EntityFrameworkCore

---

```
public class ApplicationDbContext : IdentityDbContext
{
    public ApplicationDbContext(DbContextOptions<ApplicationDbContext> options)
        : base(options)
    {
    }
}
```



# Добавление свойств пользователя

```
public class ApplicationUser : IdentityUser
{
    public byte[] Image { get; set; }
}
```

*в классе Program:*

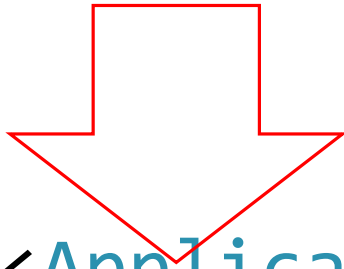
```
services.AddDefaultIdentity<ApplicationUser>
```

*в классе ApplicationDbContext:*

```
public class ApplicationDbContext :  
    IdentityDbContext<ApplicationUser>
```

В файле \_LoginPartial.cshtml

```
@inject SignInManager<IdentityUser> SignInManager  
@inject UserManager<IdentityUser> UserManager
```



```
@inject SignInManager<ApplicationUser> SignInManager  
@inject UserManager<ApplicationUser> UserManager
```

## Package Manager Console

Package source: All



Default project: WebApplication7

Package Manager Console Host Version 5.9.0.7134

Type 'get-help NuGet' to see all available NuGet commands.

PM> add-migration UserImageAdded

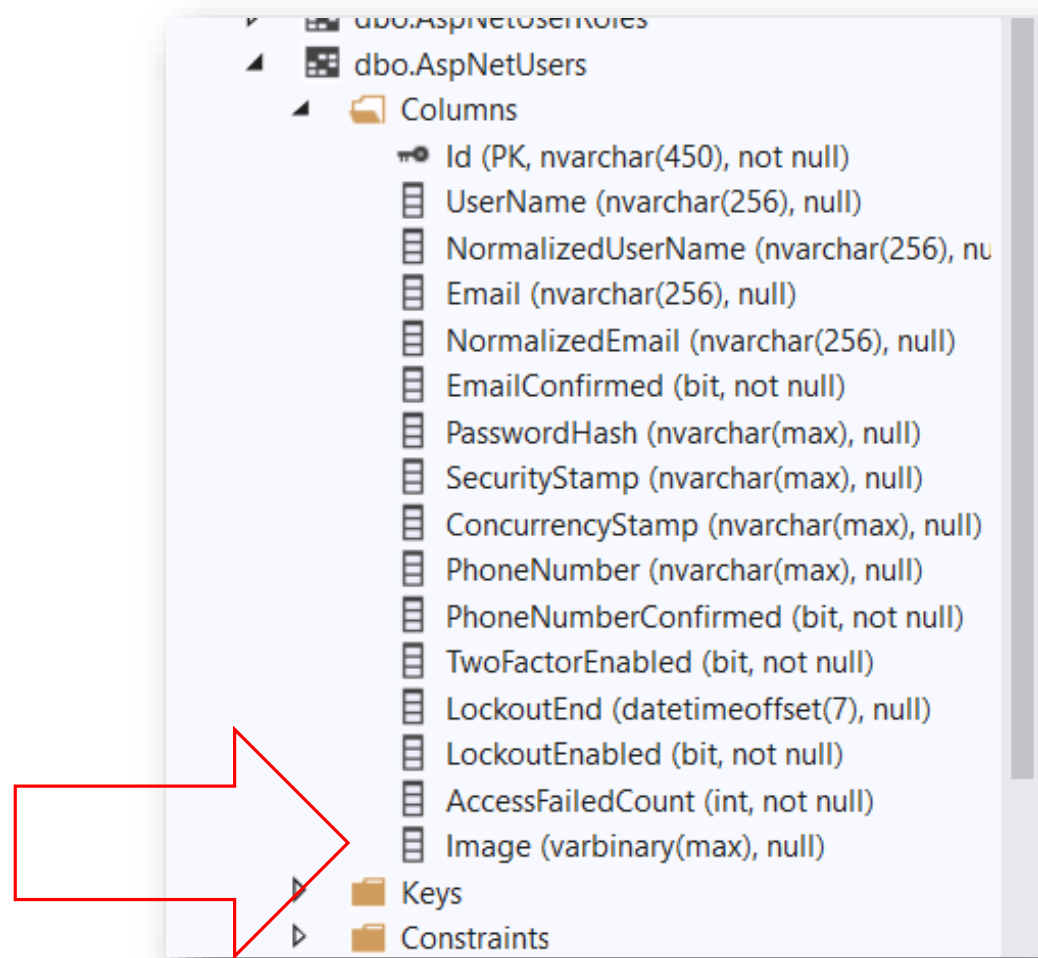
124 %



Package Manager Console

Error List

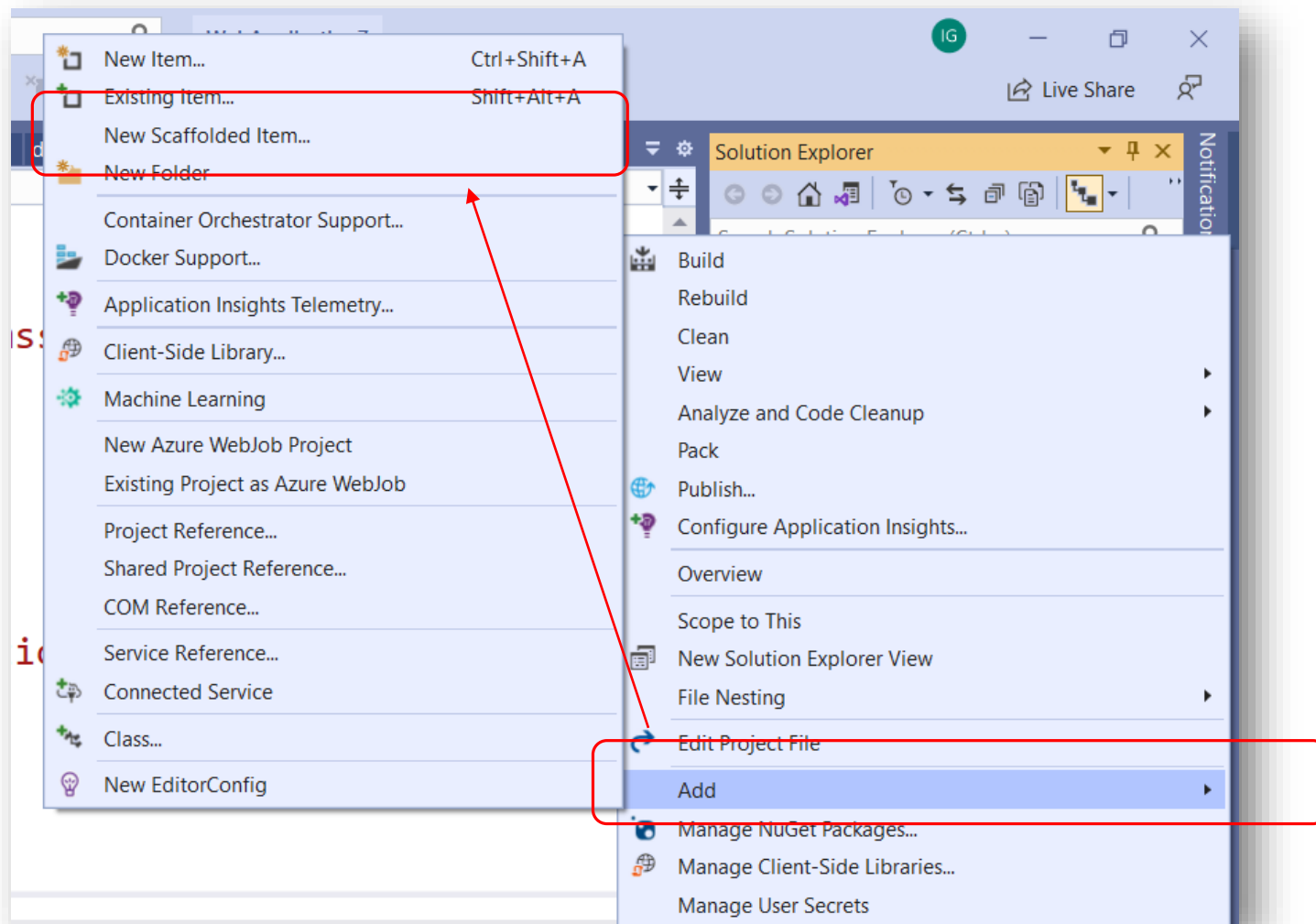
Output



# Создание страниц аутентификации

---

IDENTITY



## Add New Scaffolded Item

### Installed

#### Common

API

MVC

Razor Component

Razor Pages

**Identity**

Layout



Identity

#### Identity

by Microsoft  
v1.0.0.0

Adds code required for using ASP.NET Core Identity in the application.

Id: IdentityScaffolder

Add

Cancel



## Add Identity

Select an existing layout page, or specify a new one:

/Areas/Identity/Pages/Account/Manage/\_Layout.cshtml



(Leave empty if it is set in a Razor \_viewstart file)

☐ Override all files

Choose files to override

- |  |   |   |
|--|---|---|
| <input type="checkbox"/> Account\StatusMessage               | <input type="checkbox"/> Account\AccessDenied                   | <input type="checkbox"/> Account\ConfirmEmail               |
| <input type="checkbox"/> Account\ConfirmEmailChange          | <input type="checkbox"/> Account\ExternalLogin                  | <input type="checkbox"/> Account\ForgotPassword             |
| <input type="checkbox"/> Account\ForgotPasswordConfirmation  | <input type="checkbox"/> Account\Lockout                        | <input checked="" type="checkbox"/> Account>Login           |
| <input type="checkbox"/> Account>LoginWith2fa                | <input type="checkbox"/> Account>LoginWithRecoveryCode          | <input checked="" type="checkbox"/> Account\Logout          |
| <input type="checkbox"/> Account\Manage\Layout               | <input type="checkbox"/> Account\Manage\ManageNav               | <input type="checkbox"/> Account\Manage\StatusMessage       |
| <input type="checkbox"/> Account\Manage\ChangePassword       | <input type="checkbox"/> Account\Manage\DeletePersonalData      | <input type="checkbox"/> Account\Manage\Disable2fa          |
| <input type="checkbox"/> Account\Manage\DownloadPersonalData | <input type="checkbox"/> Account\Manage\Email                   | <input type="checkbox"/> Account\Manage\EnableAuthenticator |
| <input type="checkbox"/> Account\Manage\ExternalLogins       | <input type="checkbox"/> Account\Manage\GenerateRecoveryCodes   | <input type="checkbox"/> Account\Manage\Index               |
| <input type="checkbox"/> Account\Manage\PersonalData         | <input type="checkbox"/> Account\Manage\ResetAuthenticator      | <input type="checkbox"/> Account\Manage\SetPassword         |
| <input type="checkbox"/> Account\Manage\ShowRecoveryCodes    | <input type="checkbox"/> Account\Manage\TwoFactorAuthentication | <input checked="" type="checkbox"/> Account\Register        |
| <input type="checkbox"/> Account\RegisterConfirmation        | <input type="checkbox"/> Account\ResendEmailConfirmation        | <input type="checkbox"/> Account\ResetPassword              |
| <input type="checkbox"/> Account\ResetPasswordConfirmation   |   |   |

Data context class

ApplicationDbContext (WebApplication7.Data) +

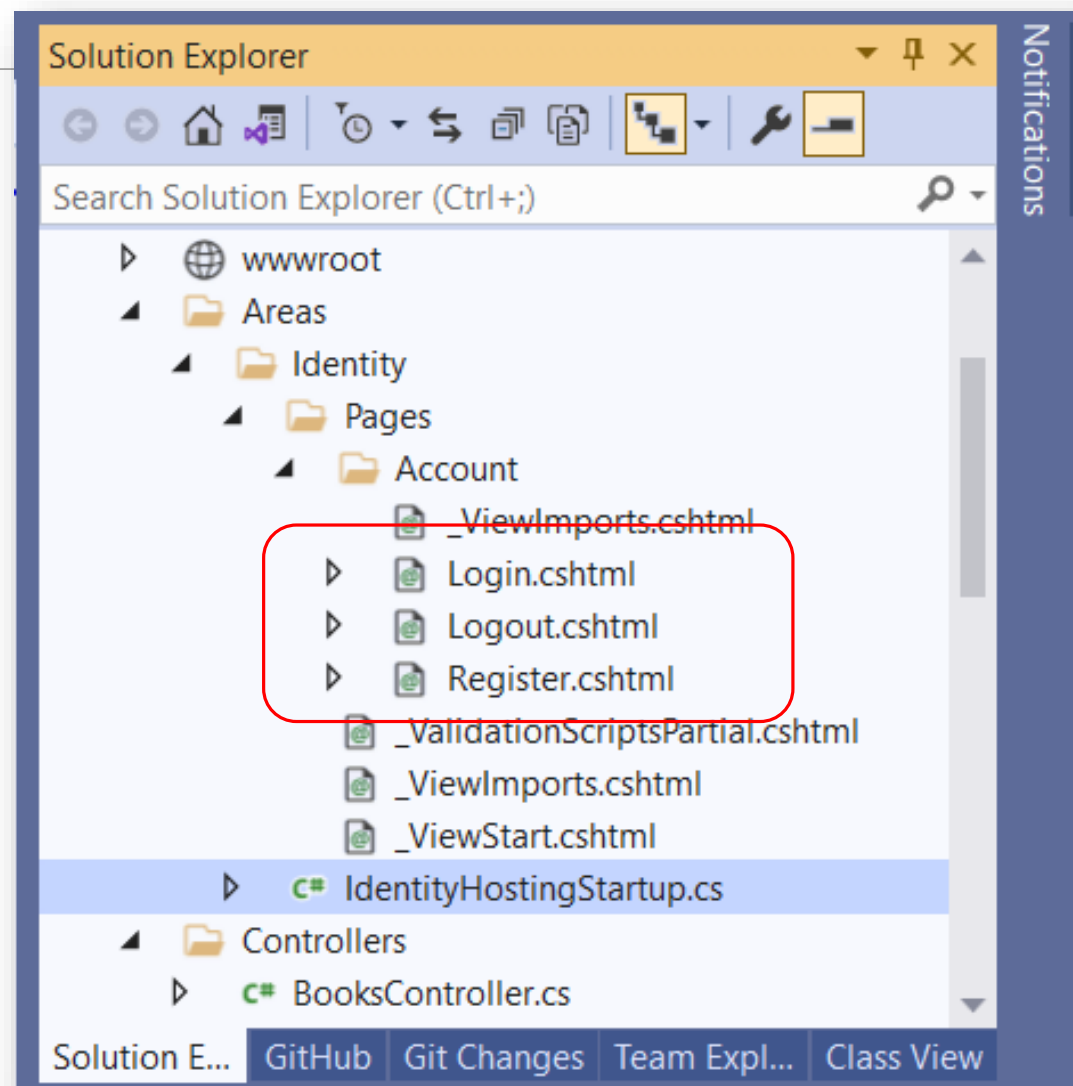
☐ Use SQLite instead of SQL Server

User class

+

Add

Cancel



# Регистрация нового пользователя

```
var user = new IdentityUser(); CreateUser();

    . . .

var result = await _userManager.CreateAsync(user, Input.Password);
if (result.Succeeded)
{
    . . .

    await _signInManager.SignInAsync(user, isPersistent: false);
    return LocalRedirect(returnUrl);
}
foreach (var error in result.Errors)
{
    ModelState.AddModelError(string.Empty, error.Description);
}

    . . .
```

# Создание роли

---

```
var roleAdmin = new IdentityRole
{
    Name = "admin",
    NormalizedName = "admin"
};

// создать роль admin
await roleManager.CreateAsync(roleAdmin);
```

# Назначение роли пользователю

---

```
adminUser = await userManager  
            .FindByEmailAsync("admin@mail.ru");  
  
await userManager.AddToRoleAsync(adminUser, "admin");
```

# Добавление Claim

---

```
var newClaim = new Claim("Department", "marketing");  
var result = await _userManager.AddClaimAsync(newUser, newClaim);
```

# Авторизация

---

АУТЕНТИФИКАЦИЯ И АВТОРИЗАЦИЯ

# Авторизация

---

```
[Authorize]
public IActionResult ShowCompanies(string[] company)
{
    return View();
}
```



# Авторизация

---

```
[Authorize(Roles = "admin, poweruser")]  
public IActionResult ShowCompanies(string[] company)  
{  
    return View();  
}
```

# Добавление политики авторизации

---

```
services.AddAuthorization(options =>
    options.AddPolicy("DepartmentCheck", policy =>
        policy.RequireClaim("Department",
            "marketing", "advertizing")));
```

# Использование политики

---

```
[Authorize(Policy = "DepartmentCheck")]  
public IActionResult ShowCompanies(string[] company)  
    {  
        return View();  
    }
```

# Регистрация через другие учетные записи

---

АУТЕНТИФИКАЦИЯ И АВТОРИЗАЦИЯ

Подключение сервиса описано в:

<https://developers.google.com/identity/sign-in/web/sign-in>

<https://console.developers.google.com/apis/credentials>



## APIs &amp; Services



Dashboard



Library



Credentials



OAuth consent screen



Domain verification



Page usage agreements

## Credentials

[+ CREATE CREDENTIALS](#)

DELETE

Create credentials to access Google APIs



Remember to

## API key

Identifies your project using a simple API key to check quota and access Google APIs

## OAuth client ID

Requests user consent so your app can access the user's data

## Service account

Enables server-to-server, app-level authentication using robot accounts

## Help me choose

Asks a few questions to help you decide which type of credential to use

## API Keys



Name



Browser

## OAuth 2.0 Client IDs



Name

Creation date

Type



## APIs &amp; Services



Dashboard



Library



Credentials



OAuth consent screen



Domain verification



Page usage agreements



## Create OAuth client ID

A client ID is used to identify a single app to Google's OAuth servers. If your app runs on multiple platforms, each will need its own client ID. See [Setting up OAuth 2.0](#) for more information.



To create an OAuth client ID, you must first set a product name on the consent screen

[CONFIGURE CONSENT SCREEN](#)

[Browse](#)

Installed

Updates **2**

## NuGet Package Manager: WebApplication7

microsoft.aspnetcore.authentication.google



Include prerelease

Package source: nuget.org



**Microsoft.AspNetCore.Authentication.Google** by Microsoft, **42,6M** down v5.0.4

ASP.NET Core contains middleware to support Google's OpenId and OAuth 2.0 authentication workflows.



**TheIdentityHub.AspNetCore.Authentication** by U2U Consult NV/SA, **7,18K** d v3.0.0

ASP.NET Core Library for The Identity Hub. The Identity Hub makes it easy to connect your app to all major identity providers like Microsoft, Facebook, Google, Twitter, Linke...



**Zitadel** by Christoph Bühler, **650** downloads v2.0.0

This is the dotnet library for authentication and authorization for <https://zitadel.ch>. The goal of the library is to

Each package is licensed to you by its owner. NuGet is not responsible for, nor does it grant any licenses to, third-party packages.



Do not show this again



**Microsoft.AspNetCore** nuget.org

Version: Latest stable 5.0.4

Install



Options

### Description

ASP.NET Core contains middleware to support Google's OpenId and OAuth 2.0 authentication workflows.

This package was built from the source code at <https://github.com/dotnet/aspnetcore/tree/e882394a7bb38930da748291fe2c2ceaa6a80588>

Version: 5.0.4

Author(s): Microsoft



---

```
services.AddAuthentication().AddGoogle(options =>
{
    options.ClientId = "xxx";
    options.ClientSecret = "xxx";
});
```

В файле appsettings.json

---

```
"Authentication": {  
  "Google": {  
    "ClientId": "xxx",  
    "ClientSecret": "yyy"  
  }  
},
```

```
services.AddAuthentication().AddGoogle(options =>
{
    IConfigurationSection googleAuthNSection =
        Configuration.GetSection("Authentication:Google");

    options.ClientId = googleAuthNSection["ClientId"];
    options.ClientSecret = googleAuthNSection["ClientSecret"];
});
```

# Создание JWT токена вручную

---

# Создание JWT токена вручную

```
[Route("api/[controller]")]
[ApiController]
public class TokenController : ControllerBase
{
    public async Task<IActionResult> GetToken(string name, string password)
    {
        // найти пользователя с введенными учетными данными
        if(user != null)
        {
            var claims = new[] {
                new Claim(JwtRegisteredClaimNames.Sub, _configuration["Jwt:Subject"]),
                new Claim(JwtRegisteredClaimNames.Jti, Guid.NewGuid().ToString()),
                new Claim(JwtRegisteredClaimNames.Iat, DateTime.UtcNow.ToString()),
                new Claim("UserId", user.Id.ToString()), new Claim("UserName", user.UserName),
                new Claim("Email", user.Email)
            };
            var key = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(_configuration["Jwt:Key"]));
            var signIn = new SigningCredentials(key, SecurityAlgorithms.HmacSha256);
            var token = new JwtSecurityToken(
                _configuration["Jwt:Issuer"], _configuration["Jwt:Audience"],
                claims,
                expires: DateTime.UtcNow.AddMinutes(10), signingCredentials: signIn);

            return Ok(new JwtSecurityTokenHandler().WriteToken(token));
        }
        return BadRequest();
    }
}
```

# Создание JWT токена вручную

```
var claims = new[] {  
    new Claim(JwtRegisteredClaimNames.Sub, _configuration["Jwt:Subject"]),  
    new Claim(JwtRegisteredClaimNames.Jti, Guid.NewGuid().ToString()),  
    new Claim(JwtRegisteredClaimNames.Iat, DateTime.UtcNow.ToString()),  
    new Claim("UserId", user.Id.ToString()),  
    new Claim("UserName", user.UserName),  
    new Claim("Email", user.Email)  
};  
  
var key = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(_configuration["Jwt:Key"]));  
var signIn = new SigningCredentials(key, SecurityAlgorithms.HmacSha256);  
var token = new JwtSecurityToken(  
    _configuration["Jwt:Issuer"], _configuration["Jwt:Audience"],  
    claims,  
    expires: DateTime.UtcNow.AddMinutes(10), signingCredentials: signIn);  
  
return Ok(new JwtSecurityTokenHandler().WriteToken(token));
```