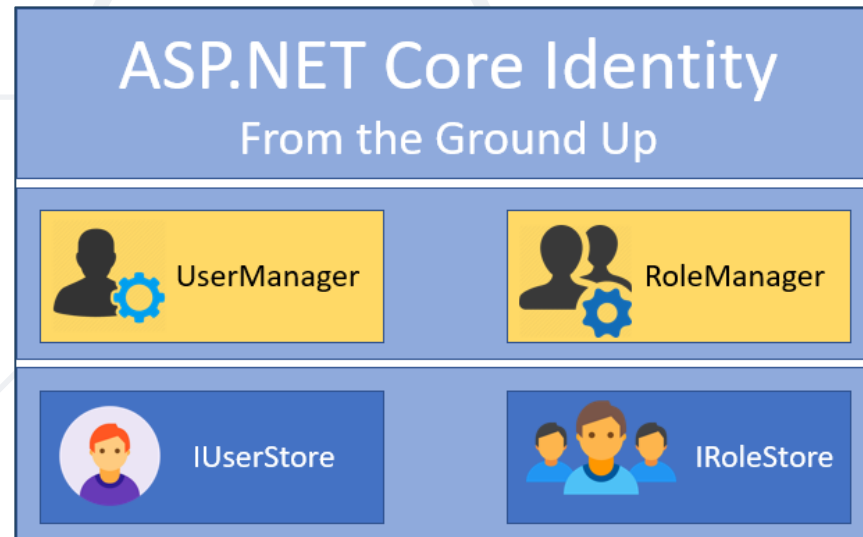


Identity

ASP.NET Core Identity, Authentication Types, Social Accounts, JWT



SoftUni Team
Technical Trainers



SoftUni

Software University

<https://softuni.bg>

sli.do

#csharp-web

Table of Contents

1. Gaining full control over Identity
2. Claims
3. Roles
4. Authentication Types
5. Social Accounts
6. JWT (JSON Web Tokens)





Gaining Full Control Over Identity

- **ApplicationUser.cs** – can add user functionality
- Extends the **user** information from the ASP.NET Core **IdentityUser**
- May hold **additional fields**
 - E.g., first name, last name, birthday

```
public class ApplicationUser
    : IdentityUser
{
    public string FirstName { get; set; }
    public string LastName { get; set; }
}
```

```
...public class IdentityUser<TKey> where TKey : IEquatable<TKey>
{
    ...public IdentityUser();
    ...public IdentityUser(string userName);

    ...public virtual DateTimeOffset? LockoutEnd { get; set; }
    ...public virtual bool TwoFactorEnabled { get; set; }
    ...public virtual bool PhoneNumberConfirmed { get; set; }
    ...public virtual string PhoneNumber { get; set; }
    ...public virtual string ConcurrencyStamp { get; set; }
    ...public virtual string SecurityStamp { get; set; }
    ...public virtual string PasswordHash { get; set; }
    ...public virtual bool EmailConfirmed { get; set; }
    ...public virtual string NormalizedEmail { get; set; }
    ...public virtual string Email { get; set; }
    ...public virtual string NormalizedUserName { get; set; }
    ...public virtual string UserName { get; set; }
    ...public virtual TKey Id { get; set; }
    ...public virtual bool LockoutEnabled { get; set; }
    ...public virtual int AccessFailedCount { get; set; }

    ...public override string ToString();
}
```

- The default identity behavior is replaced with a custom one

```
builder.Services.AddIdentity<IdentityUser, IdentityRole>()  
    // services.AddDefaultIdentity<IdentityUser>()  
    .AddEntityFrameworkStores<ApplicationDbContext>()  
    .AddDefaultTokenProviders();  
}
```

- You implement a custom **User** and a Custom **User Role**

- The following sets the **LoginPath**, **LogoutPath**, **AccessDeniedPath**

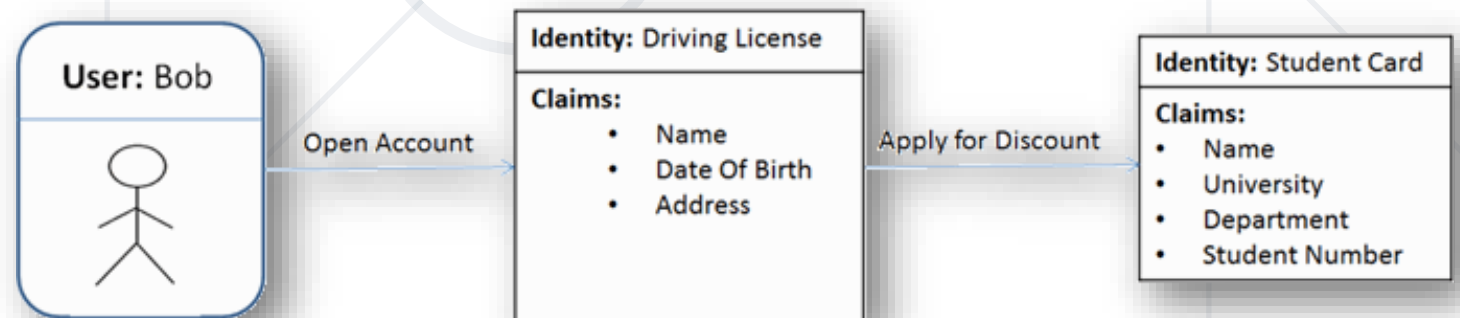
```
builder.Services.ConfigureApplicationCookie(options =>
{
    options.LoginPath = $"/Identity/Account/Login";
    options.LogoutPath = $"/Identity/Account/Logout";
    options.AccessDeniedPath = $"/Identity/Account/AccessDenied";
});
```



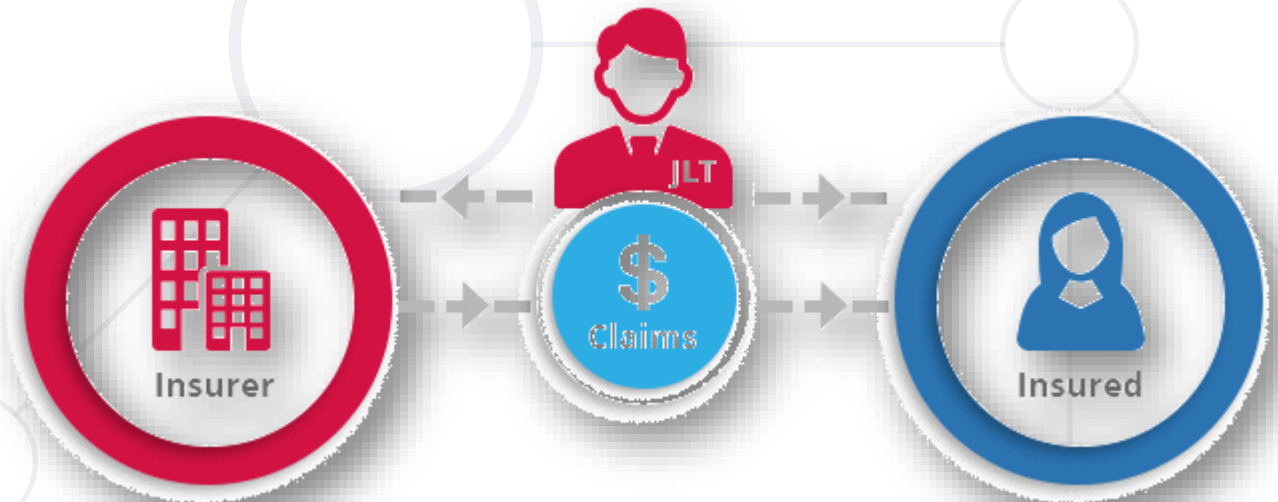
Claims

Claim-based Identity In ASP.NET Core

- **Claim**-based identity is a common technique used in applications
 - Applications acquire identity info about their users through **Claims**
- A **Claim** is a statement that one subject makes about itself
 - It can be about a name, group, ethnicity, privilege, association etc.
 - The subject making the claim is a **provider**
- **Claim**-based identity **simplifies** authentication logic
 - No **account creation / modification** required



- In **ASP.NET Core**, **Claim**-based auth checks are **declarative**
 - The developer embeds them against a **Controller** or an **Action**
 - The developer specifies **required claims** to access the functionality
- **Claims requirements** are policy based
 - The developer must register a policy expressing claims requirements
- **Claims** are **name-value** pairs



- The simplest type of **claim policy** checks only for the **presence** of a claim
 - The **value** of the **claim** is not checked

```
builder.Services.AddAuthorization(options =>
{
    options.AddPolicy("EmployeeOnly", policy =>
        policy.RequireClaim("EmployeeNumber"));
});
```

```
//This action is accessible only by Identities with the "EmployeeOnly" Claim...
[Authorize(Policy = "EmployeeOnly")]
public IActionResult VacationBalance() => View();
```



Roles

- Role-based **authorization** in ASP.NET Core
 - An identity may belong to **one** or **more roles**
- Roles are claims, but **not** all claims are roles
- Enable **RoleManager** with

```
builder.Services.AddDefaultIdentity<IdentityUser>(...)  
                .AddRoles<IdentityRole>();
```



- Adding a **User** to an **existing Role**

```
public async Task<IActionResult> AddUserToRole()
{
    var roleName = "Administrator";
    var roleExists = await roleManager.RoleExistsAsync(roleName);

    if (roleExists)
    {
        var user = await userManager.GetUserAsync(User);
        var result = await userManager.AddToRoleAsync(user, roleName);

        if (result.Succeeded)
            // The user is now Administrator
    }
}
```

Require Logged-In User in Certain Role

- Give access only to Users in Role "**Administrator**"

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

- Give access if User's Role is "**User**", "**Student**" or "**Trainer**"

```
[Authorize(Roles="User, Student, Trainer")]  
public ActionResult Roles()  
{  
    ...  
}
```

Check the Currently Logged-In User's Role

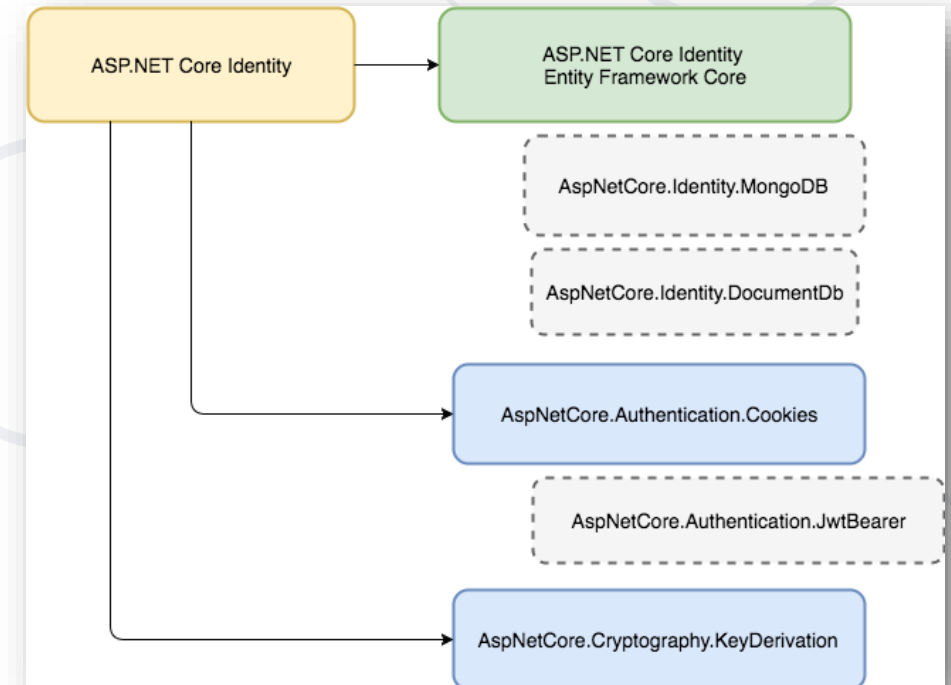
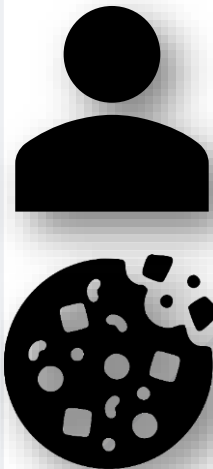
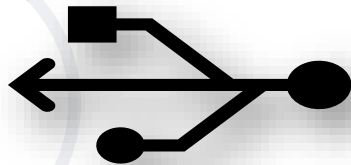
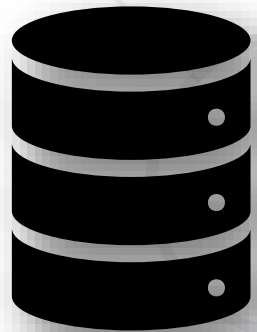
```
// GET: /Home/Admin (for logged-in admins only)  
[Authorize]  
public ActionResult Admin()  
{  
    if (this.User.IsInRole("Administrator"))  
    {  
        ViewBag.Message = "Welcome to the admin area!";  
        return View();  
    }  
  
    return this.View("Unauthorized");  
}
```



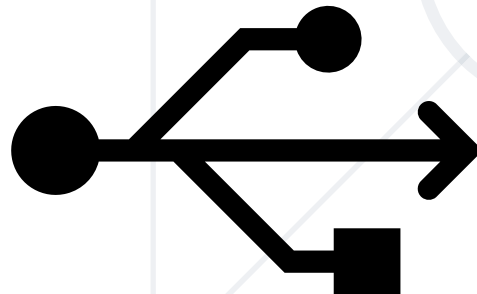

Authentication Types

- **Cookie-based** Authentication & Authorization (Identity)
 - Authentication is entirely **Cookie-based**
 - The **Principal** is based on **claims**
- **Windows** Authentication & Authorization
 - Relies on the operating system to authenticate users
- **Cloud-based** Authentication & Authorization
 - An **external platform** handles the User functionality
- **JSON Web Tokens (JWT)** Authentication & Authorization

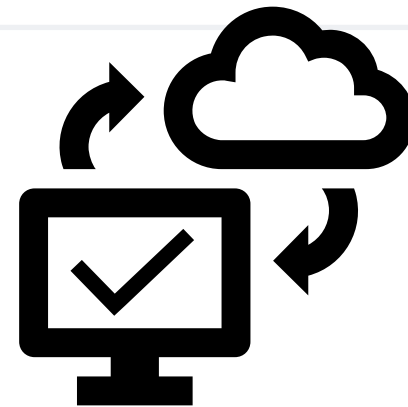
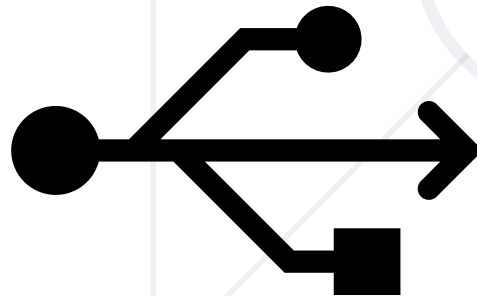
- **Cookie-Based** auth is the **ASP.NET Core** app auth mechanism
 - Authentication is entirely **Cookie-based**
 - This is a major difference from **ASP.NET MVC**
 - The **Principal** is based on **claims**



- **Windows** auth is a more complex auth mechanism
 - Relies on the operating system to authenticate users
 - Credentials are hashed before sent across the network
 - Best suited for intranet environments
 - Clients, Users, Servers belong to the same Windows domain (AD)

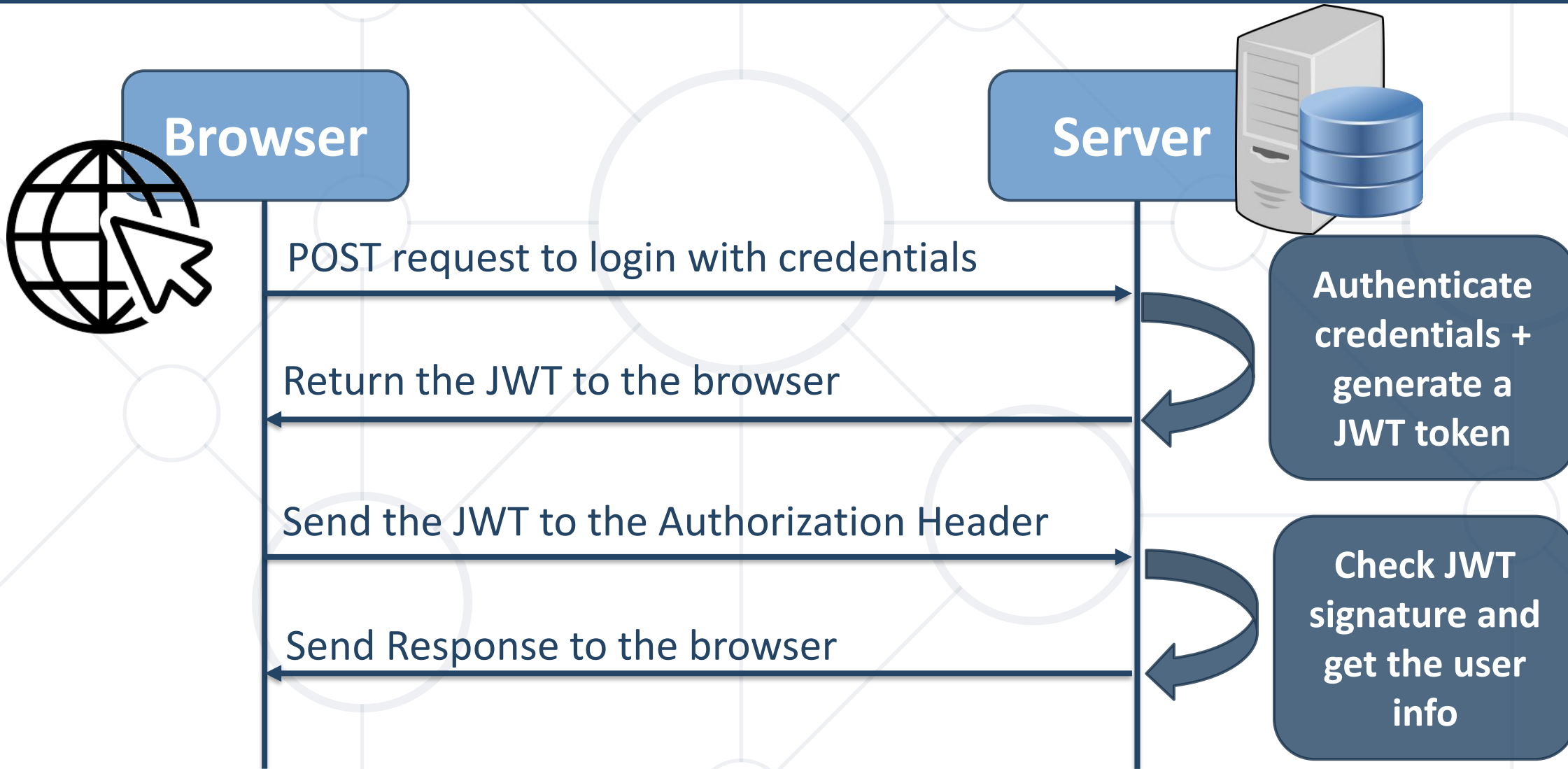


- **Cloud-based** auth is a more modern authentication approach
 - Authentication & Authorization work is outsourced
 - An **external platform** handles the User functionality
 - Ensures flexibility and speed
 - Greatly decouples the auth functionality from the others



- **JSON Web Tokens** is a modern JavaScript-based auth mechanism
 - Compact and self-contained
 - Focused on **signed tokens**
 - Work with claims
 - Data is encrypted
 - Used for auth & information exchange
 - Commonly used, when developing **REST**
 - Extremely simple to comprehend

JWT Authentication & Authorization



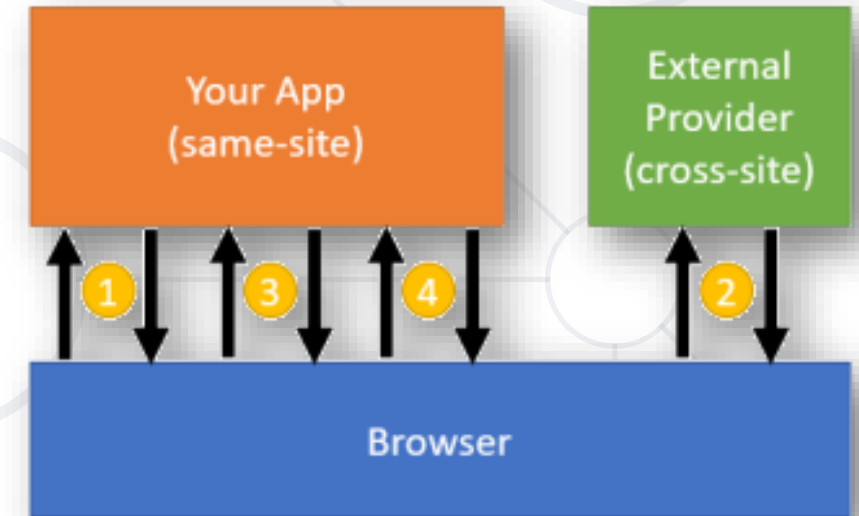


Social Accounts


- Enabling users to sign in with their existing credentials is convenient
 - Shifts the complexities of managing the sign-in process to third party
 - Enhances user experience by minimizing their auth activities
- **ASP.NET Core** supports built-in external login providers for
 - Google
 - Facebook
 - Twitter
 - Microsoft

```
builder.Services.AddAuthentication()  
    .AddGoogle(googleOptions => { ... })  
    .AddFacebook(facebookOptions => { ... })  
    .AddTwitter(twitterOptions => { ... })  
    .AddMicrosoftAccount(microsoftOptions => { ... });
```

- Each **External Login provider** has some Developer API
 - You have to configure an application there before using it
 - That application will provide you with credentials
 - Application **ID** + Application **Secret**
- These credentials will be used by the external provider API
 - You authenticate yourself with them, when sending a request
 - They should **not** be stored in the **open world**



External Login Provider Developer API

facebook DEVELOPERS [Documentation](#) [Support](#) [Blog](#) [Apps](#)  Pat Patterson ▾

Settings ▸

- Basic
- Auth Dialog
- Advanced

[Open Graph](#)

[Roles](#)


[Credits](#)


[Insights](#)

Related links

- [Use Debug Tool](#)
- [Use Graph API Explorer](#)
- [See App Timeline View](#)
- [Promote with an Ad](#)
- [Translate your App](#)
- [Delete App](#)

Apps ▸ Force.com Toolkit App v3 ▸ Basic



Force.com Toolkit App v3
App ID: 357898090905558
App Secret: 42835e36d6bf1eb127ec68112b91b72b (reset)
 (edit icon)

Basic Info

App Display Name: [?]

App Namespace: [?]

Contact Email: [?]

App Domain: [?]

Category: [?]

Other ▾

Choose a sub-category ▾

Cloud Services

Hosting URL: [?] You have not generated a URL through one of our partners (Get one)

Select how your app integrates with Facebook

☒ Website ✕

Site URL: [?]

☒ App on Facebook I want people to run my app inside Facebook.com.

- On the back-end, it is quite simple, and quite clean
- Example: **Facebook**

```
builder.Services.AddAuthentication()  
    .AddFacebook(facebookOptions => {  
        facebookOptions.AppId =  
            configuration["Authentication:Facebook:AppId"];  
        facebookOptions.AppSecret =  
            configuration["Authentication:Facebook:AppSecret"];  
    });
```

- If you use the **default ASP.NET Core Login** page, this will add a **form**

Configuring Social Accounts

WebApplication1 Home About Contact

Log in

Use a local account to log in.

Email

Password

☐ Remember me?

[Forgot your password?](#)

[Register as a new user](#)

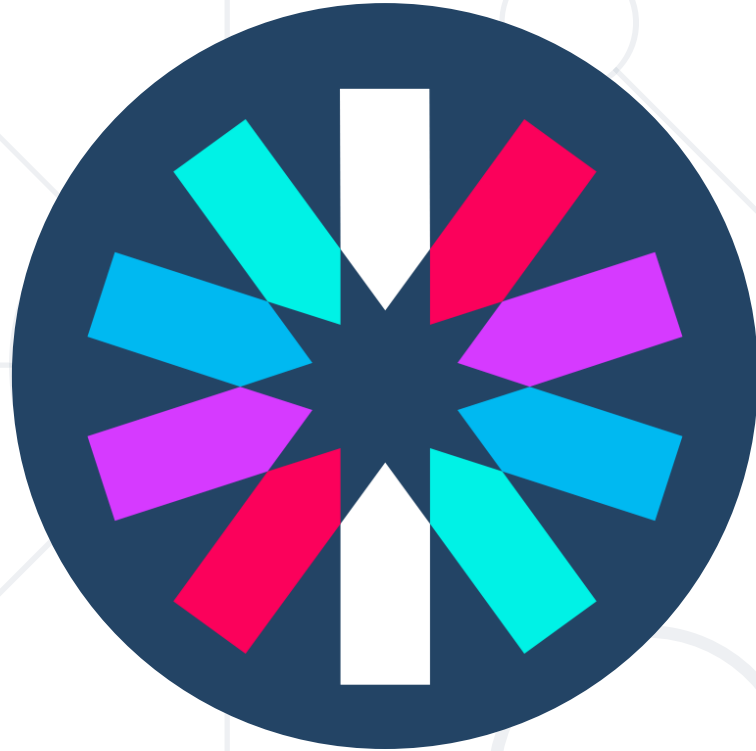
Use another service to log in.

Sends **POST** request to
/Identity/Account/ExternalLogin

Button submits a parameter:
name: **provider**
value: {**externalLogin**}

Name	Headers	Preview	Response	Cookies	Timing
<input type="checkbox"/> ExternalLogin?returnUrl=%2F/Identity/Account	▼ Form Data view source view URL encoded				
	provider: Facebook				
	__RequestVerificationToken: <input type="text"/>				

129 requests | 91.8 KB transferred | Finish: 1.0 min...



JSON Web Tokens

JWT

- **JWT** is a method for representing claims between two parties
 - An open, industry standard – RFC 7519
 - Easy to use, and at the same time – absolutely secured
- When the user successfully **authenticates** (login) using their credentials:
 - A **JSON Web Token** is generated and returned
 - It must be stored (in **local** / **session** storage, **cookies** are also an option)
- Whenever a protected route is accessed, the user agent sends the **JWT**
 - Typically in an **Authorization** header, using the **Bearer** schema

- **JWT** is absolutely **stateless**, nothing is stored on the server
- Here is an example of an encoded and decoded **JSON Web Token**

The parts of the **token** are separated by **dots**

Encoded

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.  
eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MzkwMDIyLnQsInR5cCI6IkpXVCJ9.  
Sf1KxwRJSMeKKF2QT4fwpMeJf36P0k6yJV_adQssw5c
```

As any normal auth **JWT** also has an **expiration**

The parts of the **token** are in a strict **order**

The token data does not change the **token format**

Decoded

Header: (algorithm, token type)

```
{  
  "alg": "HS256",  
  "typ": "JWT"  
}
```

Payload: (data)

```
{  
  "sub": "1234567890",  
  "name": "John Doe",  
  "iat": 1516239022  
}
```

Verify Signature

```
HMACSHA256(base64UrlEncode(H...) +  
".", base64UrlEncode(P...), key)
```


- **JWT** in **ASP.NET Core** is configured in **Program.cs**
 - Install **Microsoft.AspNetCore.Authentication.JwtBearer**

```
{  
  "JWT": {  
    "ValidAudience": "http://localhost:4200",  
    "ValidIssuer": "http://localhost:61955",  
    "Secret": "super-secret-JWT-key"  
  },  
  ...  
}
```

appsettings.json

```
var auth = builder.Services.AddAuthentication(options =>  
{  
    options.DefaultAuthenticateScheme =  
        JwtBearerDefaults.AuthenticationScheme;  
    options.DefaultChallengeScheme =  
        JwtBearerDefaults.AuthenticationScheme;  
    options.DefaultScheme =  
        JwtBearerDefaults.AuthenticationScheme;  
});
```

```
auth.AddJwtBearer(...);
```

```
// Configure DI for application services  
builder.Services.AddScoped<IUserService, UserService>();
```

On the
next slide

JWT Configuration in ASP.NET Core

```
auth.AddJwtBearer(options =>
{
    options.SaveToken = true;
    options.RequireHttpsMetadata = false;
    options.TokenValidationParameters = new TokenValidationParameters()
    {
        ValidateIssuer = true,
        ValidateAudience = true,
        ValidAudience = builder.Configuration["JWT:ValidAudience"],
        ValidIssuer = builder.Configuration["JWT:ValidIssuer"],
        IssuerSigningKey =
            new
SymmetricSecurityKey(Encoding.UTF8.GetBytes(builder.Configuration["JWT:Secret"]))
    };
});

...

// Don't forget to add app.UseAuthentication(); and app.UseAuthorization();
```

- The controller action (**Endpoint**) is kept "**thin**" to a maximum

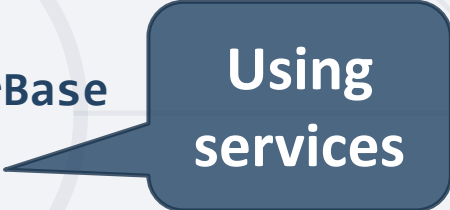
```
[ApiController]
[Route("/api/[controller]")]
public class UsersController : ControllerBase
{
    private IUserService userService;

    public UsersController(IUserService userService) => this.userService = userService;

    [HttpPost("login")]
    public IActionResult Login([FromBody] LoginUserBindingModel loginUser)
    {
        var user = this.userService.Authenticate(loginUser.Username, loginUser.Password);

        if (user == null)
            return BadRequest(new { message = "Username or password is incorrect." });

        var tokenString = this.userService.GenerateJSONWebToken(user);
        return Ok(new { token = tokenString });
    }
}
```



Using services

- Configurations for **JWT** are injected into a **service class constructor**

```
public class UserService : IUserService
{
    private AppDbContext context;
    private UserManager<IdentityUser> userManager;
    private readonly IConfiguration configuration;

    public UserService(AppDbContext context, IConfiguration configuration)
    {
        this.context = context;
        this.configuration = configuration;
        ...
    }

    public IdentityUser Authenticate(string username, string password) { ... }
    public string GenerateJSONWebToken(IdentityUser user) { ... }
}
```

On the
next slides

```
public IdentityUser Authenticate(string username, string password)
{
    var user = this.userManager.FindByNameAsync(username).Result;
    if (user != null && this.userManager.CheckPasswordAsync(user, password).Result)
        return user;

    return null;
}

public string GenerateJSONWebToken(IdentityUser user)
{
    var authClaims = new List<Claim>
    {
        new Claim(ClaimTypes.Name, user.UserName),
        new Claim(JwtRegisteredClaimNames.Jti, Guid.NewGuid().ToString()),
    };

    // Continues on the next slide...
```

```
string jwtSecret = this.configuration["JWT:Secret"];
byte[] jwtSecretBytes = Encoding.UTF8.GetBytes(jwtSecret);
var authSigningKey = new SymmetricSecurityKey(jwtSecretBytes);

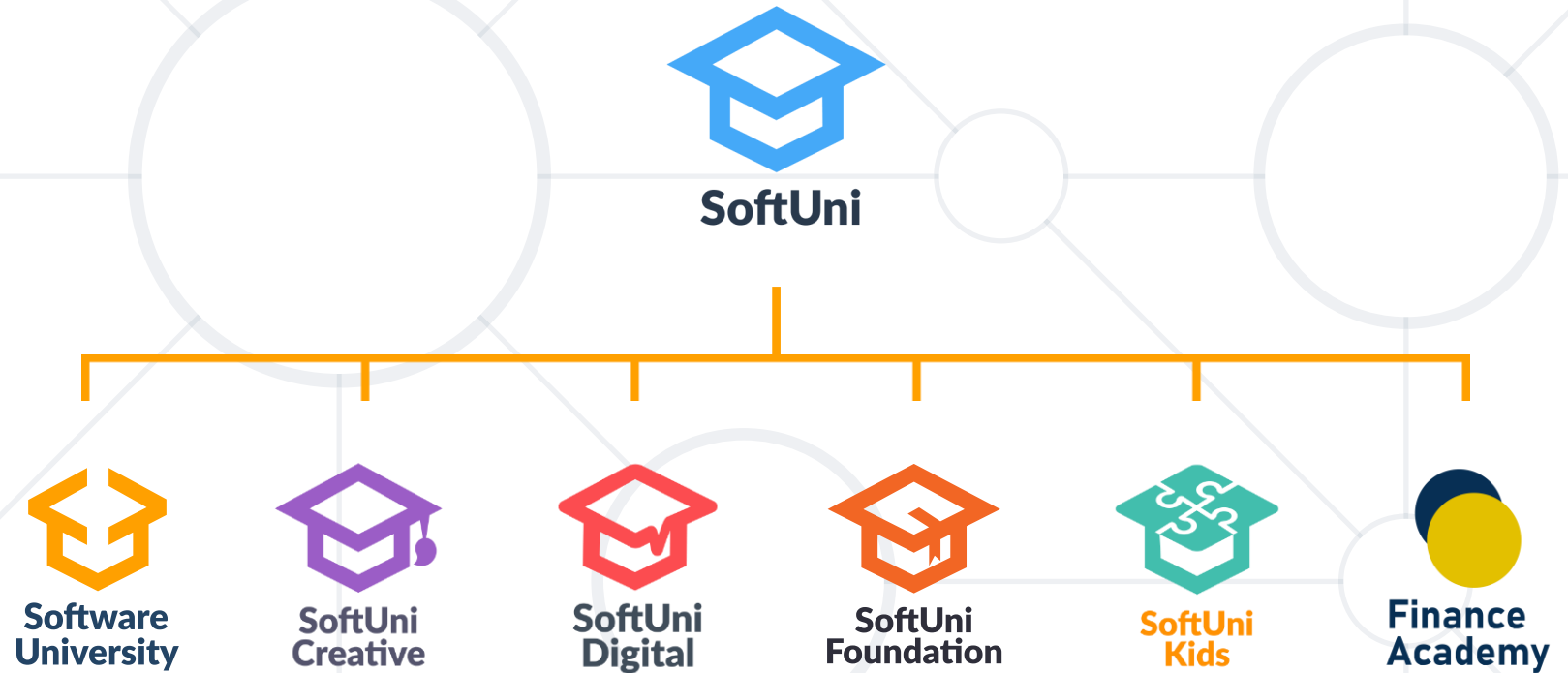
var token = new JwtSecurityToken(
    issuer: this.configuration["JWT:ValidIssuer"],
    audience: this.configuration["JWT:ValidAudience"],
    expires: DateTime.Now.AddHours(3),
    claims: authClaims,
    signingCredentials:
        new SigningCredentials(authSigningKey, SecurityAlgorithms.HmacSha256));

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

- You can gain **full control** over Identity
- **Claims** == a statement that a user makes about themselves
- **Roles** are required to gain access to a specific resource
- There are different **authentication** types
- **Social Accounts** enable users to sign in with their existing credentials
- **JWT** =method for representing claims between two parties



Questions?



SoftUni Diamond Partners



- Software University – High-Quality Education, Profession and Job for Software Developers
 - softuni.bg, softuni.org
- Software University Foundation
 - softuni.foundation
- Software University @ Facebook
 - facebook.com/SoftwareUniversity



- This course (slides, examples, demos, exercises, homework, documents, videos and other assets) is **copyrighted content**
- Unauthorized copy, reproduction or use is illegal
- © SoftUni – <https://softuni.org>
- © Software University – <https://softuni.bg>

