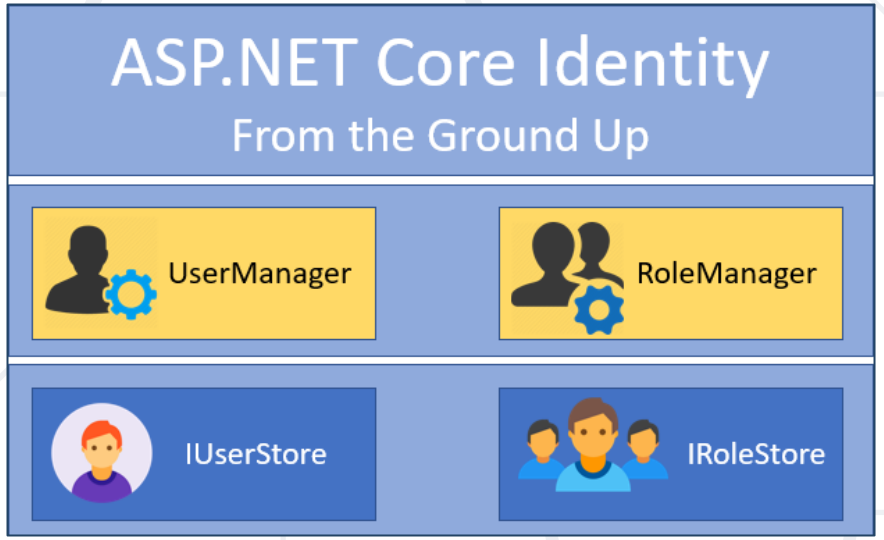
# Identity

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




**SoftUni Team**

**Technical Trainers**

Software University

SoftUni

Software University

https://softuni.bg

# Have a Question?

Software University

sli.do

#csharp-web

# Table of Contents



# Gaining Full Control Over Identity

# Extending ASP.NET Core 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();
}
```

# Gaining Full Identity Control

- 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**

# Gaining Full Identity Control

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

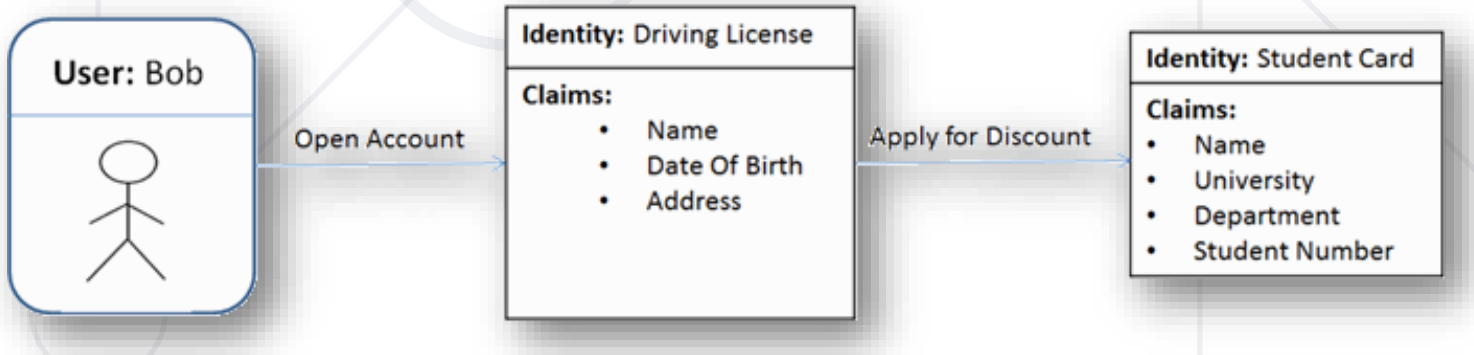
```
builder.Services.ConfigureApplicationCookie(options =>
{
        options.LoginPath = $"/Identity/Account/Login";
        options.LogoutPath = $"/Identity/Account/Logout";
        options.AccessDeniedPath = $"/Identity/Account/AccessDenied";
});
```

# Claims

## Claim-based Identity In ASP.NET Core

# Claims

- **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



# Claims in ASP.NET Core

- 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

# Claims

- 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"));
});
```

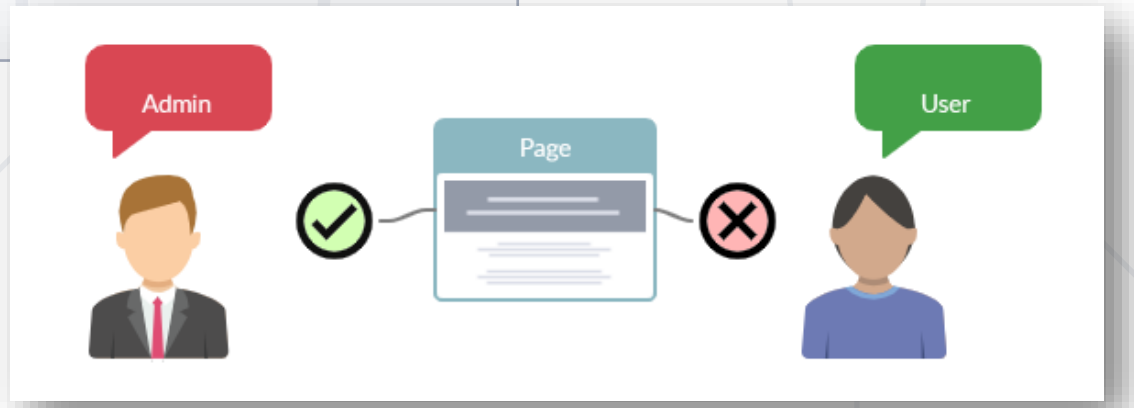
```
//This action is accessible only by Identities with the "EmployeeOnly" Claim...
[Authorize(Policy = "EmployeeOnly")]
public IActionResult VacationBalance() => View();
```

# Roles

# 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>();
```



# Add User to a Role

- 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

Software University

```
// 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

# Authentication Types in ASP.NET Core

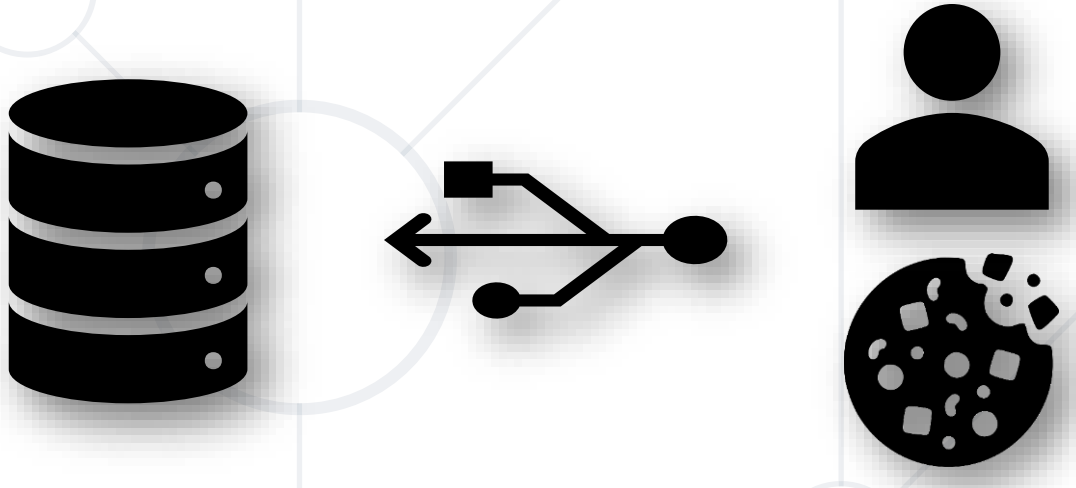


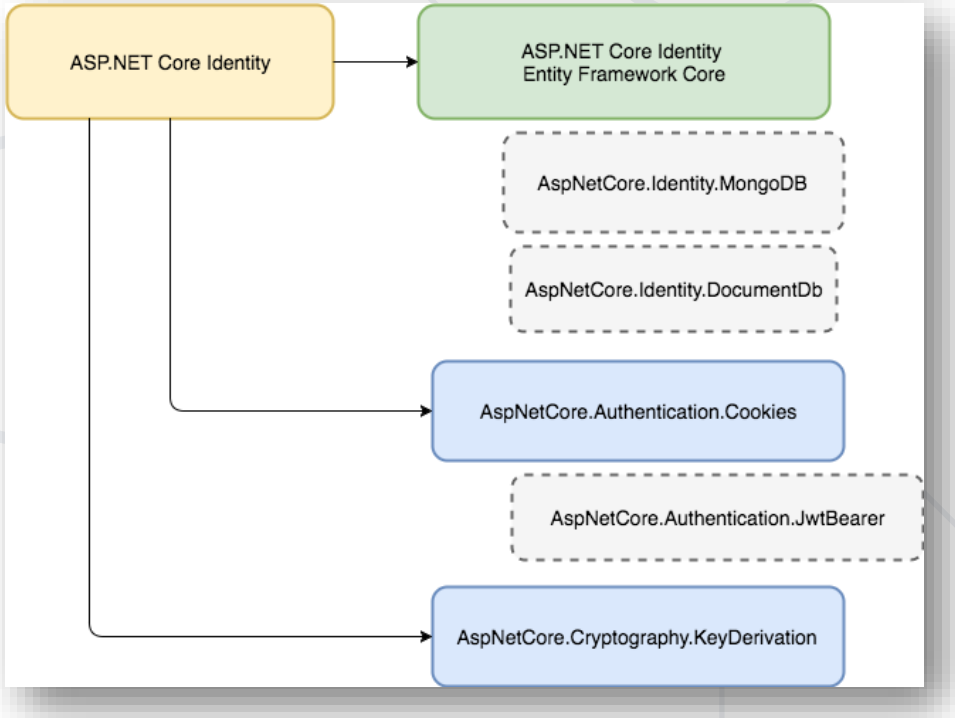
- **Cookie-based** Authentication & Authorization (Identity)
  - Authentication is entirely **Cookie-based**
  - The **Principal** is based on **claims**
- **Windows** Authentication & Authorization
  - Relies on the operating system to authenticate users
- **Cloud-based** Authentication & Authorization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

# Cookie-Based Authentication & Authorization

Software University

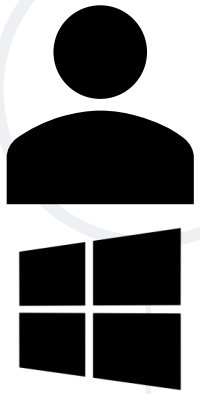
- **Cookie-Based** auth is the **ASP.NET Core** app auth mechanism
  - Authentication is entirely **Cookie-based**
  - This is a major difference from **ASP.NET MVC**
  - The **Principal** is based on **claims**

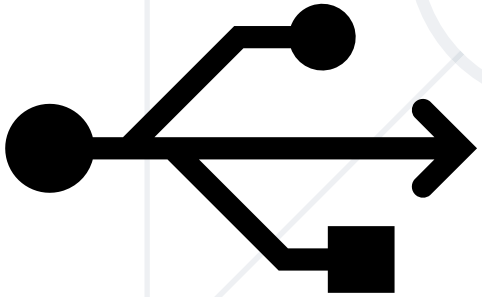




# Windows Authentication & Authorization

- **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)

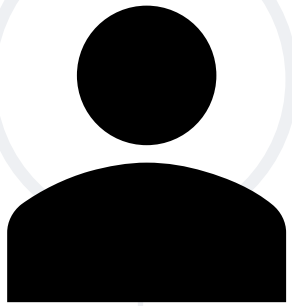


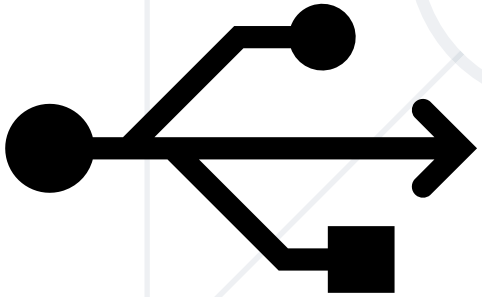


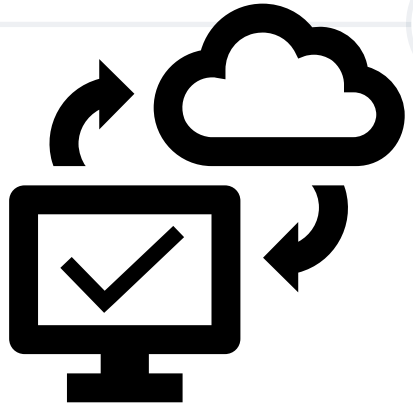
# Cloud-based Authentication & Authorization

Software University

- **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





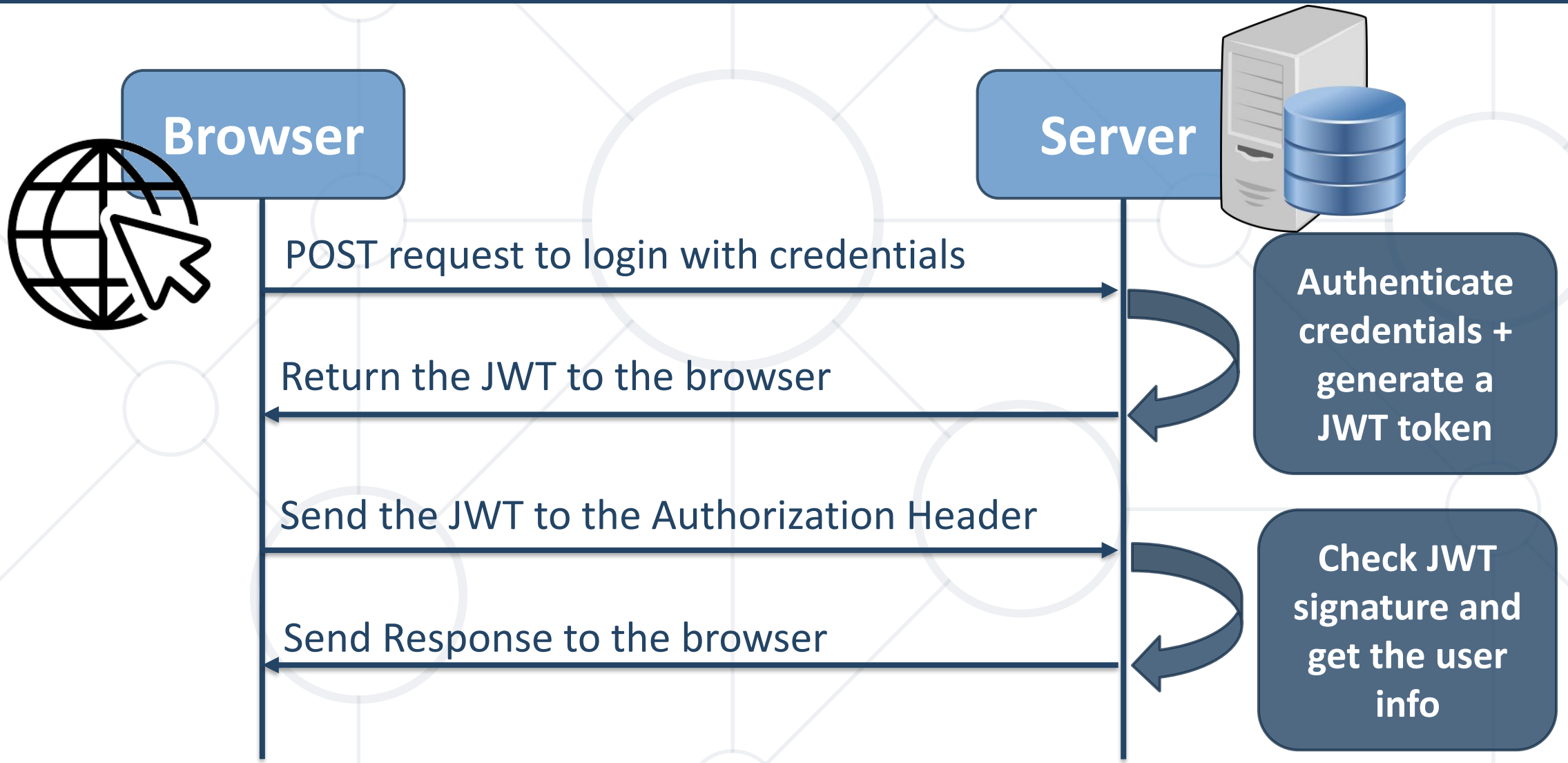


# JWT Authentication & Authorization

Software University

- **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

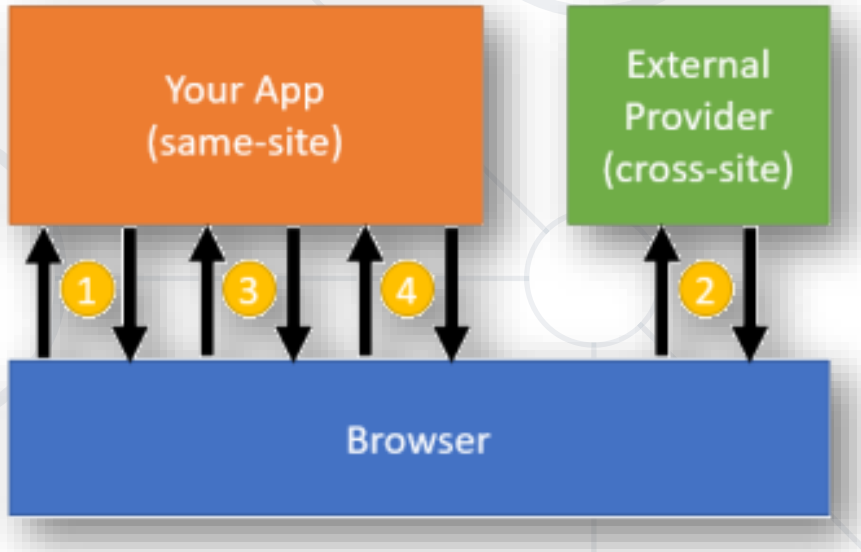
# Social Accounts

- 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

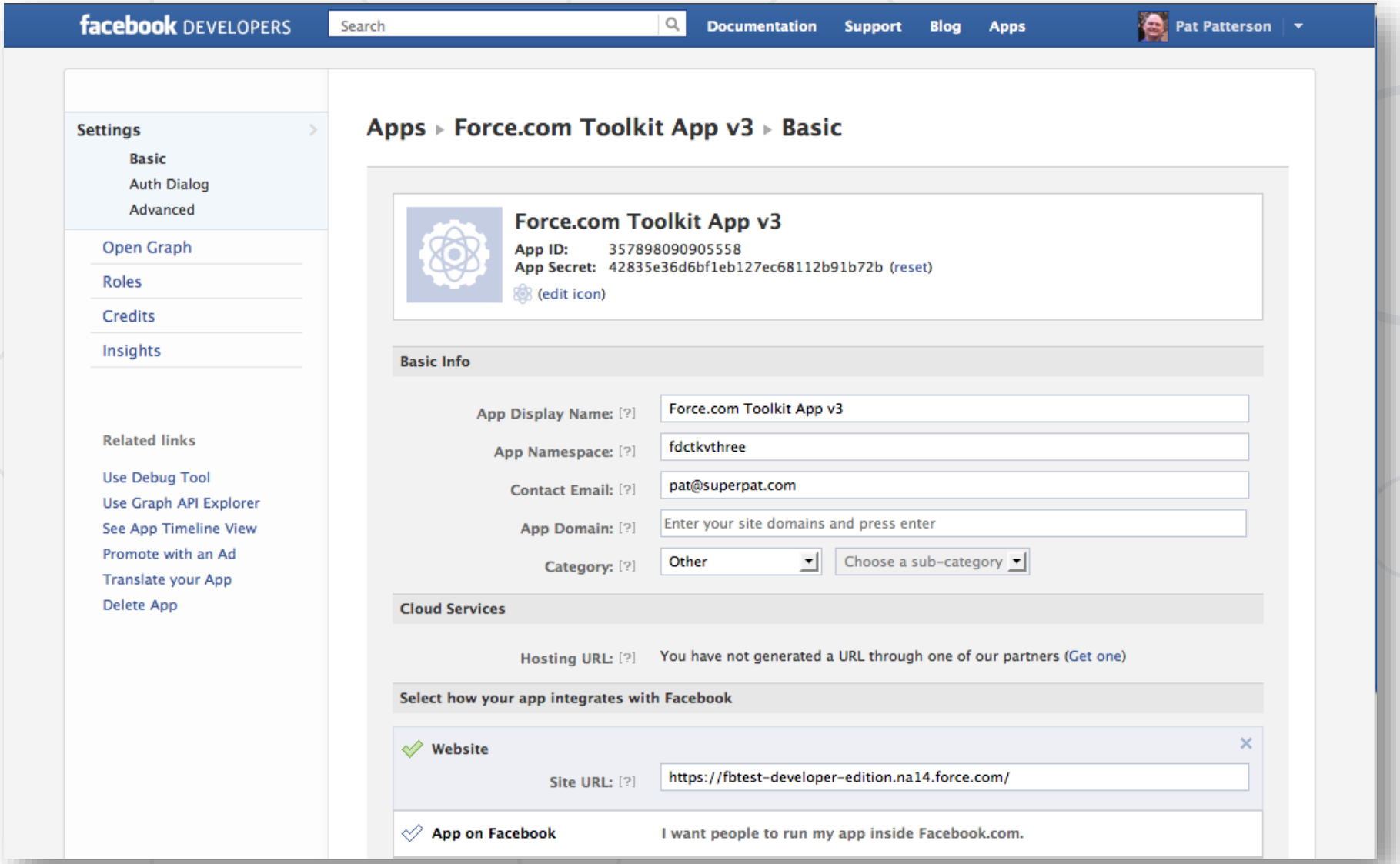
```
builder.Services.AddAuthentication()
        .AddGoogle(googleOptions => { ... })
        .AddFacebook(facebookOptions => { ... })
        .AddTwitter(twitterOptions => { ... })
        .AddMicrosoftAccount(microsoftOptions => { ... });
```

# External Login Provider

- 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



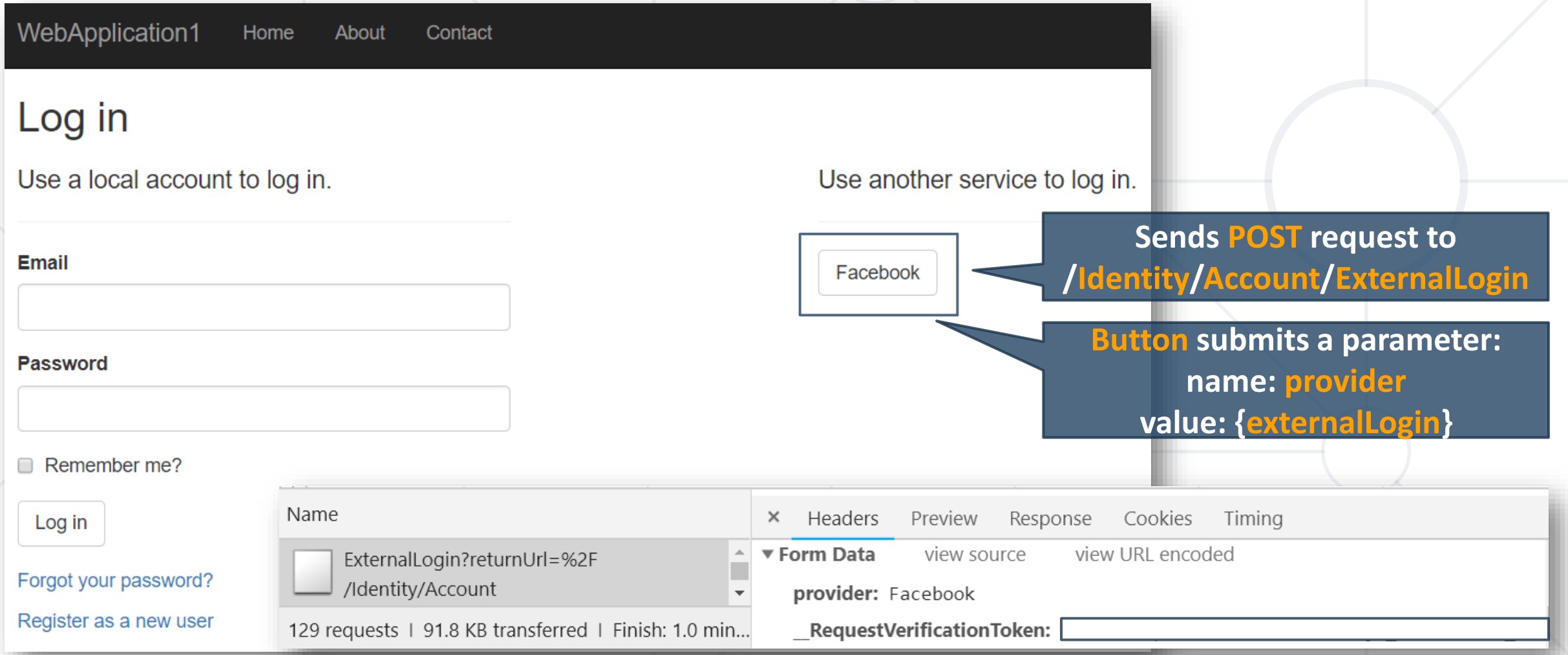
# Configuring Social Accounts

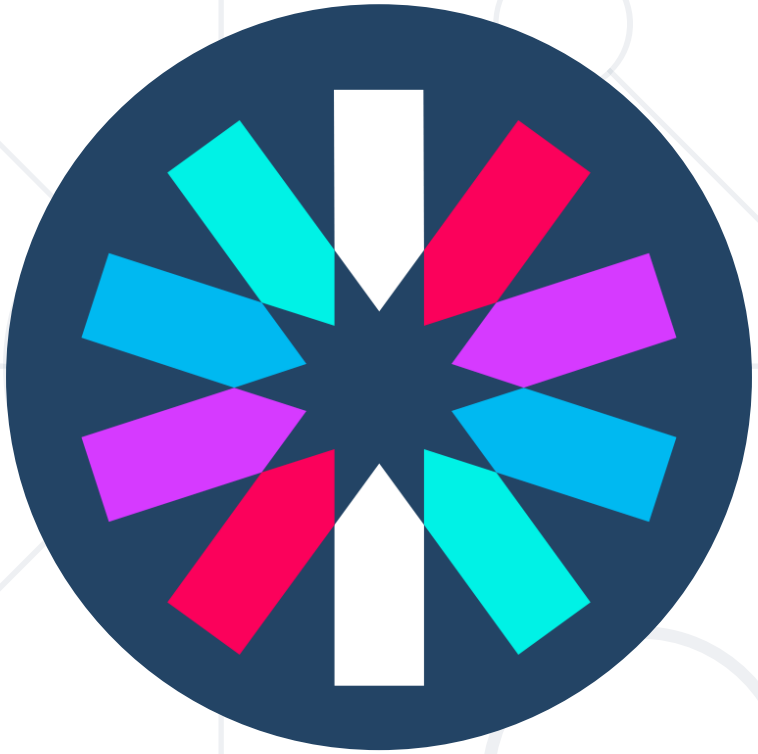
- On the back-end, it is quite simple, and quite clean
- Example: **Facebook**

```
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





# JSON Web Tokens

JWT

# JSON Web Tokens

- **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

# JSON Web Tokens

- **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**

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

**Encoded**

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9
.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI
6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDI
yfQ.SflKxwRJSMeKKF2QT4fwpMeJf36POk6y
JV_adQssw5c
```

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 Configuration in ASP.NET Core

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

appsettings.json

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

```
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

Software University

```
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();
```

# JWT in ASP.NET Core Controllers

- 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

# JWT in ASP.NET Core in 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

# JWT in ASP.NET Core in Services

Software University

```
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...
```

# JWT in ASP.NET Core in Services

Software University

```
    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);
}
```

# Summary

- 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

# SoftUni Diamond Partners

Software University

SmartIT

Coca-Cola HBC Bulgaria

SUPER HOSTING .BG

createX

Postbank

Решения за твоето утре

DRAFT KINGS

THE CROWN IS YOURS

VIVACOM

AMBITIONED

INDEAVR

Serving the high achievers

PHAR VISION

# Questions?

SoftUni

Software University

SoftUni Creative

SoftUni Digital

SoftUni Foundation

SoftUni Kids

СофтУни БУДИТЕЛ

Finance Academy

# Trainings @ Software University (SoftUni)

- 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

# License

- 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