ASP.NET Core
Auth*

Антон Марченко, 2020

Auth*

- Identification (who you are)
- Authentication (prove that it is you)
- Authorization (what you’re allowed)

Пример auth* - Закрытый клуб

- Идентификация – спрашивают имя и фамилию
- Аутентификация – просят показать паспорт
- Авторизация – проверка по списку гостей

Виды аутентификации

- По паролю username/password
- По сертификатам
- По одноразовым паролям
- По ключам доступа
- По токенам

Аутентификации по паролю

- HTTP authentication
- Forms authentication
- URL query
- Request body
- HTTP header

Аутентификация

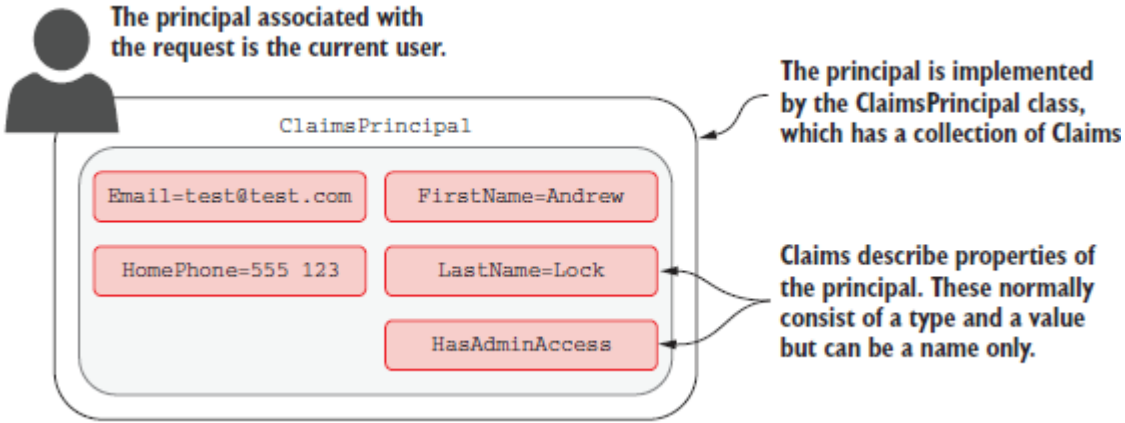
- Principal - контекст безопасности, в котором выполняется код
- Хранится в HttpContext.User

User

- Principal запроса - текущий пользователь приложения
- Реализуется ClaimsPrincipal

ClaimsPrincipal

- Свойства Claim ключ-значение
- Частички информации о пользователе



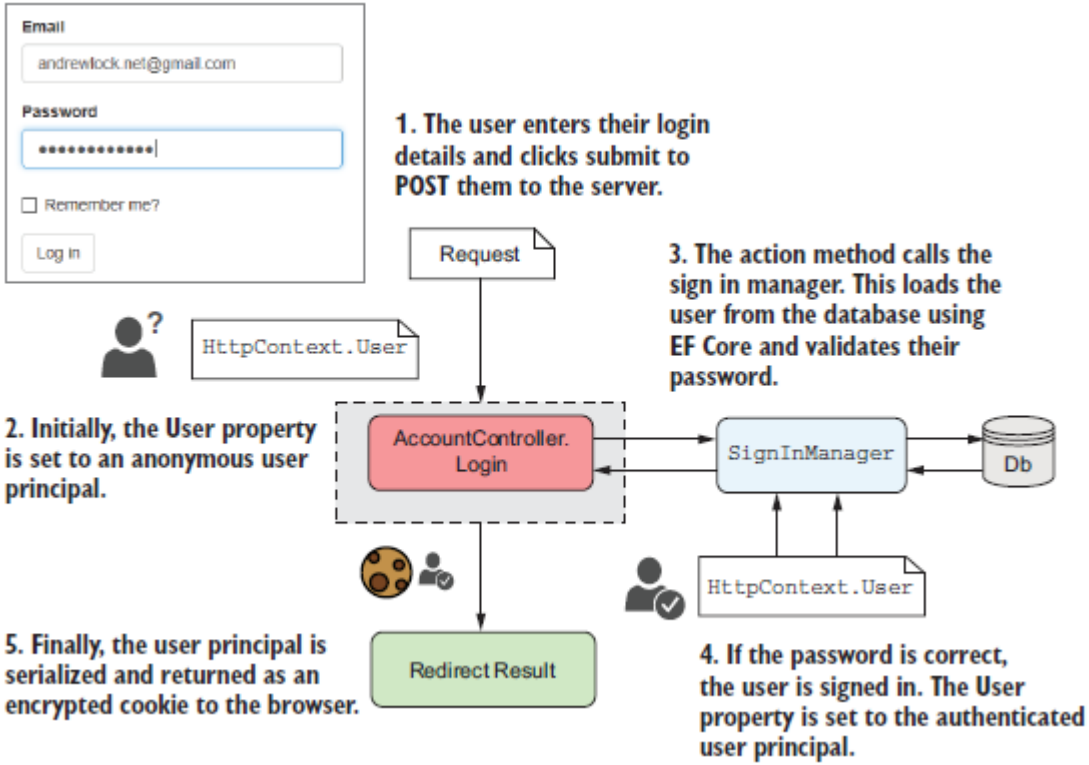
Claims vs Roles approach

- Подход с ролями - устаревший
- Claims поддерживает его для обратной совместимости

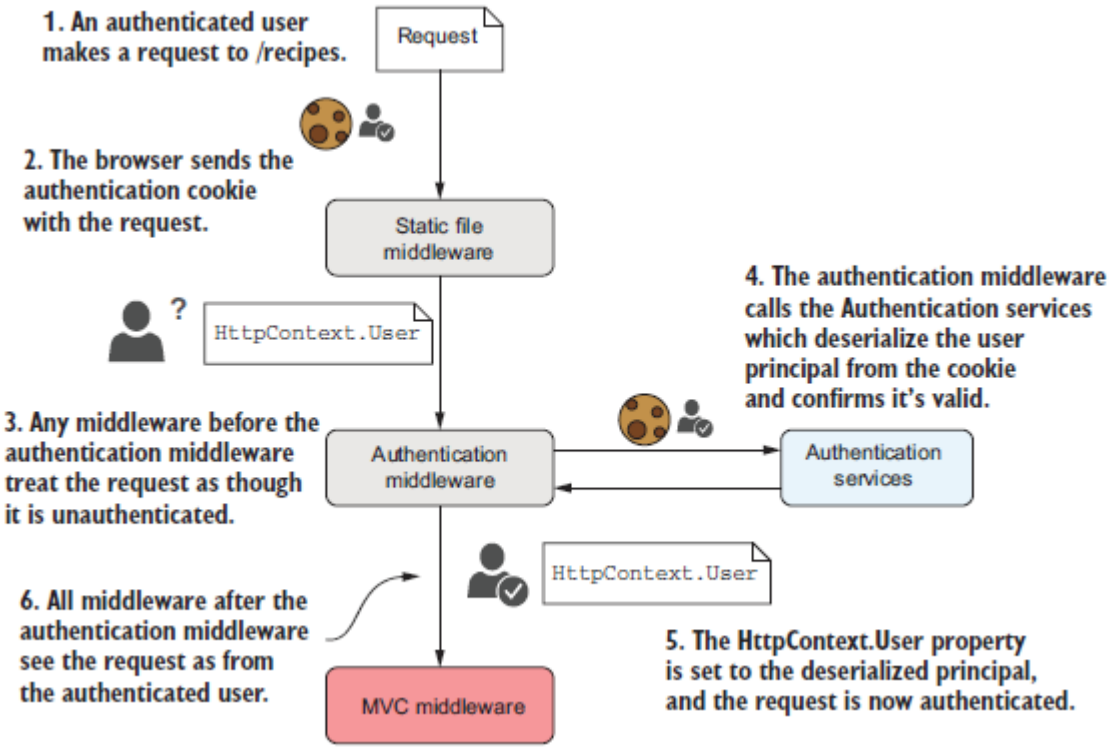
Аутентификация

- Клиент посылает логин и пароль
- Приложение верифицирует логин находит пользователя, проверяет пароль
- Устанавливается зашифрованный ClaimsPrincipal в cookie

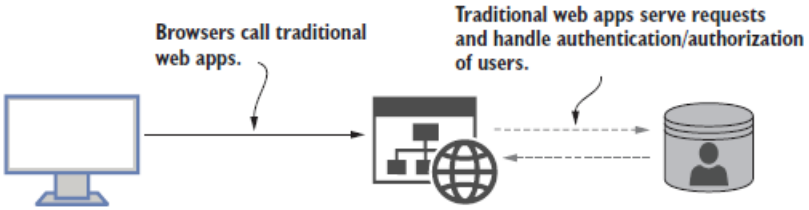
Вход в приложение

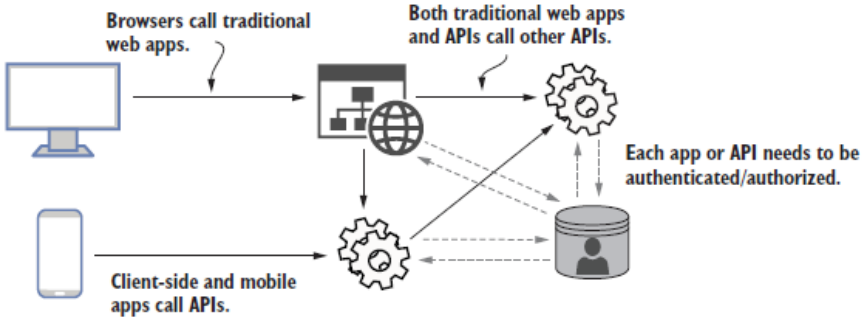


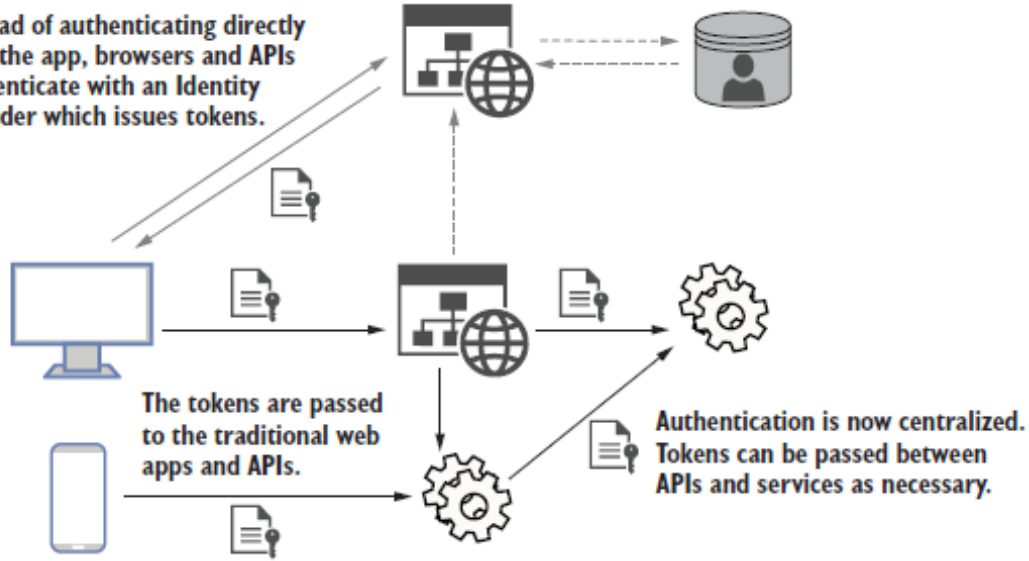
Middleware


1. An authenticated user makes a request to /recipes.
Request
2. The browser sends the authentication cookie with the request.
Static file middleware
?
HttpContext.User
4. The authentication middleware calls the Authentication services which deserialize the user principal from the cookie and confirms it's valid.
3. Any middleware before the authentication middleware treat the request as though it is unauthenticated.
Authentication middleware
Authentication services
HttpContext.User
6. All middleware after the authentication middleware see the request as from the authenticated user.
5. The HttpContext.User property is set to the deserialized principal, and the request is now authenticated.
MVC middleware

SPA&Mobile authentication







Identity providers

- OpenIdDict
- AspNet.Security.OpenIdConnect.Server
- IdentityServer4

Веб-приложение - менеджер учетных записей, генерирует токены

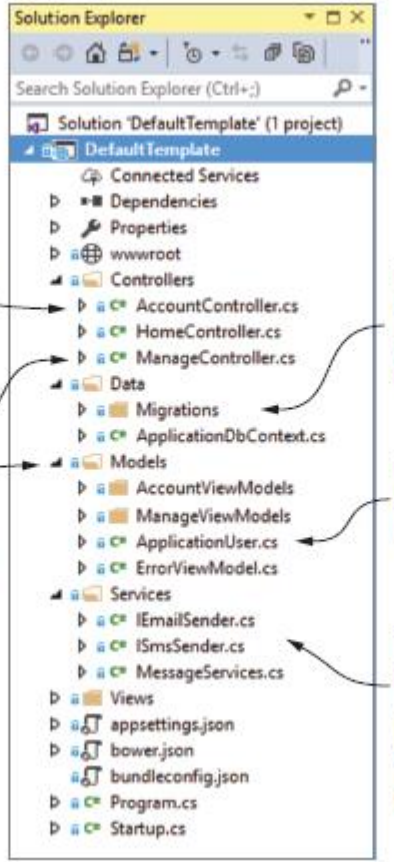
ASP.Net Core Identity

- Система управления пользователями
  - Сервисы для регистрации и управления учетными записями пользователей
  - Интеграция с EF Core
  - Работа с паролями - хранение и валидация
  - Вход пользователей

Возможности Identity

Managed by ASP.NET Core Identity	Implemented by the developer
Database schema for storing users and claims.	UI for logging in, creating, and managing users (controller, actions, view models). This is included in the default templates.
Creating a user in the database.	Sending emails and SMS messages.
Password validation and rules.	Customizing claims for users (adding new claims).
Handling user account lockout (to prevent brute-force attacks).	Configuring third-party identity providers.
Managing and generating 2FA codes.	
Generating password-reset tokens.	
Saving additional claims to the database.	
Managing third-party identity providers (for example Facebook, Google, Twitter).	

Шаблон проекта с Identity



The AccountController is used to create new users and sign in.

The ManageController is used to manage your account, such as changing you password.

The Models folder contains view models for the MVC controllers.

The template includes an EF Core DbContext and migrations to configure the database schema for ASP.NET Core Identity.

The ApplicationUser is the EF Core entity and is used as the ASP.NET Core Indentity user type.

The services folder contains stub interfaces and services. These can be expanded to provide additional features such as password reset emails and two-factor authentication.

Сервисы Identity

ASP.NET Core Identity uses EF Core, so it includes the standard EF Core configuration.

Adds the Identity system, and configures the user and role types

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

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

    services.AddTransient<IEmailSender, AuthMessageSender>();
    services.AddTransient<ISmsSender, AuthMessageSender>();

    services.AddMvc();
}
```

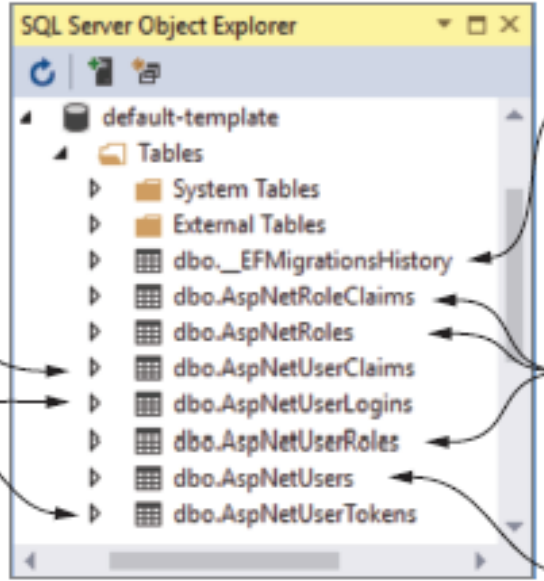
Configures Identity to store its data in EF Core

Uses the default Identity providers for generating 2FA codes

Registers the stub service for sending SMS; may be missing in some versions of Visual Studio

Registers the stub service for sending email

Identity's database schema



The claims associated with each user are stored in AspNetUserClaims.

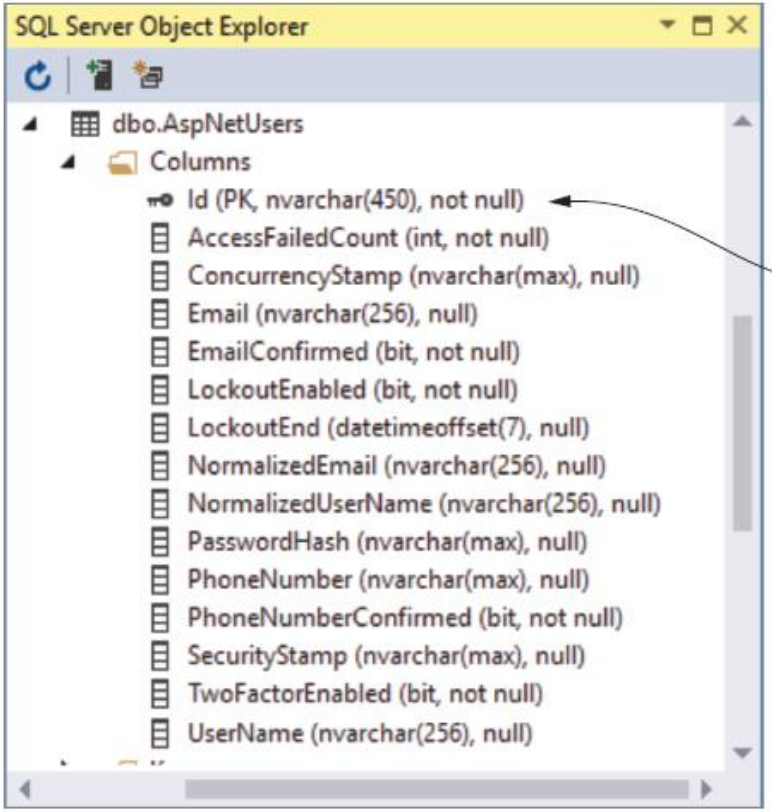
The AspNetUserLogins and AspNetUserTokens are used to manage details of third-party logins like Facebook and Google.

ASP.NET Core uses EF Core migrations. The history of applied migrations is stored in the __EFMigrationsHistory table.

The AspNetRoles, AspNetRoleClaims, and AspNetUserRoles provide role-based authorization for legacy reasons.

The user entities are stored in the AspNetUsers table.

AspNetUsers



By default, ASP.NET Core Identity uses GUIDs for the user Id stored as strings in the database.

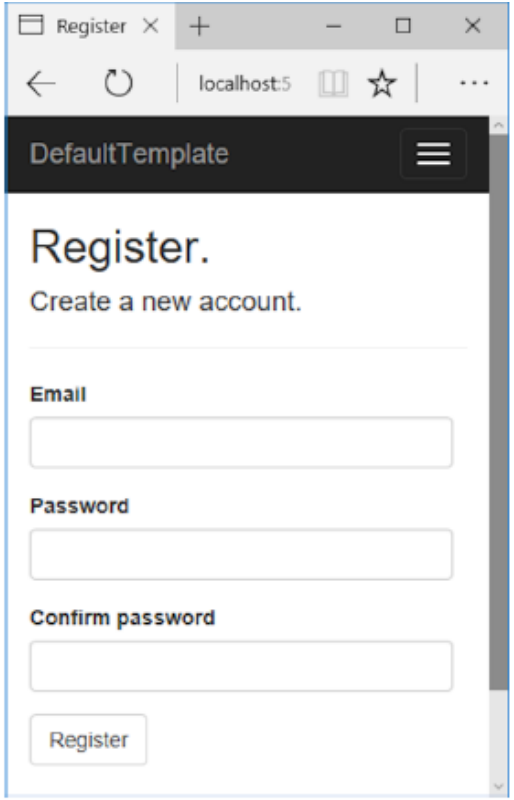
The table contains all the neccessary fields for authenticating a user, email and phone number confirmation, two factor authentication, and account lockout.

Основные классы в ASP.NET Core Identity

Контекст данных **IdentityDbContext** содержит следующие свойства:

Users: набор объектов IdentityUser, соответствует таблице пользователей
Roles: набор объектов IdentityRole, соответствует таблице ролей
RoleClaims: набор объектов IdentityRoleClaim, соответствует таблице связи ролей и объектов claims
UserLogins: набор объектов IdentityUserLogin, соответствует таблице связи пользователей с их логинами их внешних сервисов
UserClaims: набор объектов IdentityUserClaim, соответствует таблице связи пользователей и объектов claims
UserRoles: набор объектов IdentityUserRole, соответствует таблице, которая сопоставляет пользователей и их роли
UserTokens: набор объектов IdentityUserToken, соответствует таблице токенов пользователей

Пример: action для регистрации




```
public async Task<IActionResult> Register(
        RegisterViewModel model, string returnUrl = null)
{
    ViewData["ReturnUrl"] = returnUrl;
    if (ModelState.IsValid)
    {
        var user = new ApplicationUser {
            UserName = model.Email, Email = model.Email };
        var result = await _userManager.CreateAsync(
            user, model.Password);

        if (result.Succeeded)
        {
            await _signInManager.SignInAsync(user);
            return RedirectToLocal(returnUrl);
        }
        AddErrors(result);
    }

    return View(model);
}
```

The form is bound to RegisterViewModel.

If the binding mode is valid, creates an instance of the user entity

Checks that the provided password is valid and creates the new user in the database

If the user was created successfully, sign them in.

Updates the HttpContext.User principal and sets a cookie containing the serialized principal

If something went wrong, adds the errors to the model state and redisplays the form

Redirects to the previous page using a helper method to protect against open redirect attacks

Adding custom claims

```
public async Task<IActionResult> Register(
        RegisterViewModel model, string returnUrl = null)
{
    ViewData["ReturnUrl"] = returnUrl;
    if (ModelState.IsValid)
    {
        var user = new ApplicationUser {
            UserName = model.Email, Email = model.Email };
        var result = await _userManager.CreateAsync(
            user, model.Password);

        if (result.Succeeded)
        {
            var claim = new Claim("FullName", model.Name);
            await _userManager.AddClaimAsync(user, nameClaim);

            await _signInManager.SignInAsync(user);
            return RedirectToLocal(returnUrl);
        }
        AddErrors(result);
    }

    return View(model);
}
```

Creates an instance of the ApplicationUser entity, as usual

Validates that the provided password is valid and creates the user in the database

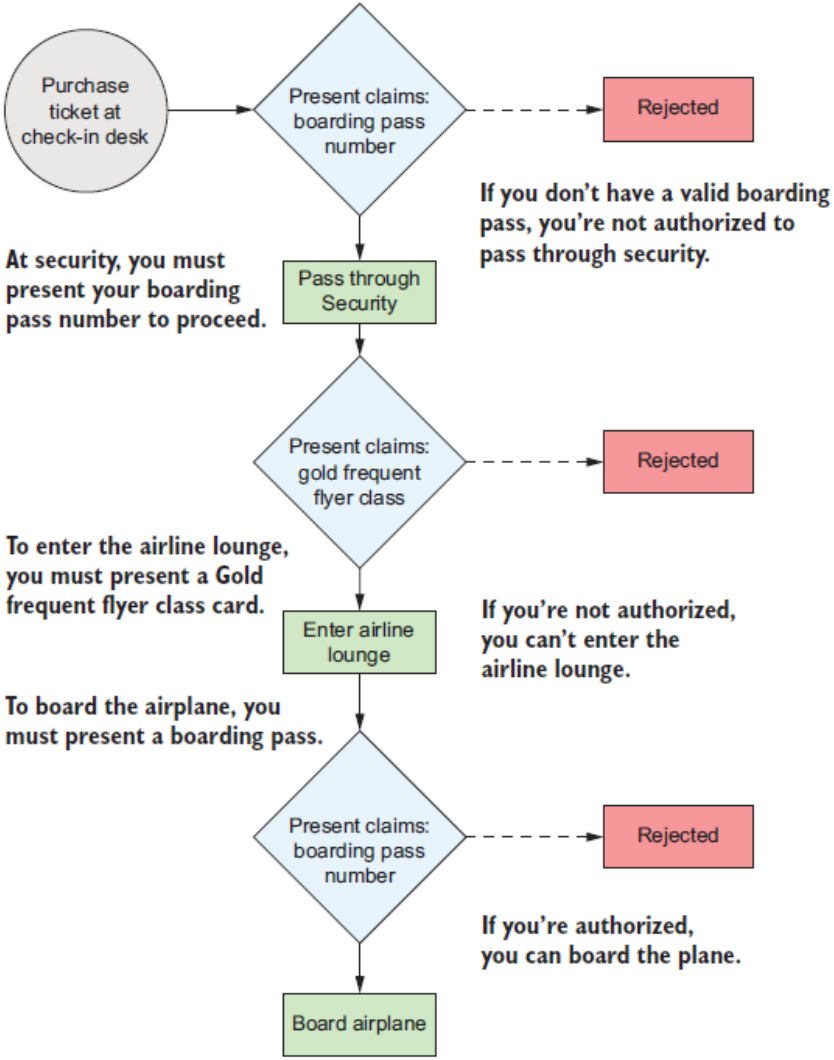
Creates a claim, with a string name of "FullName" and the provided value

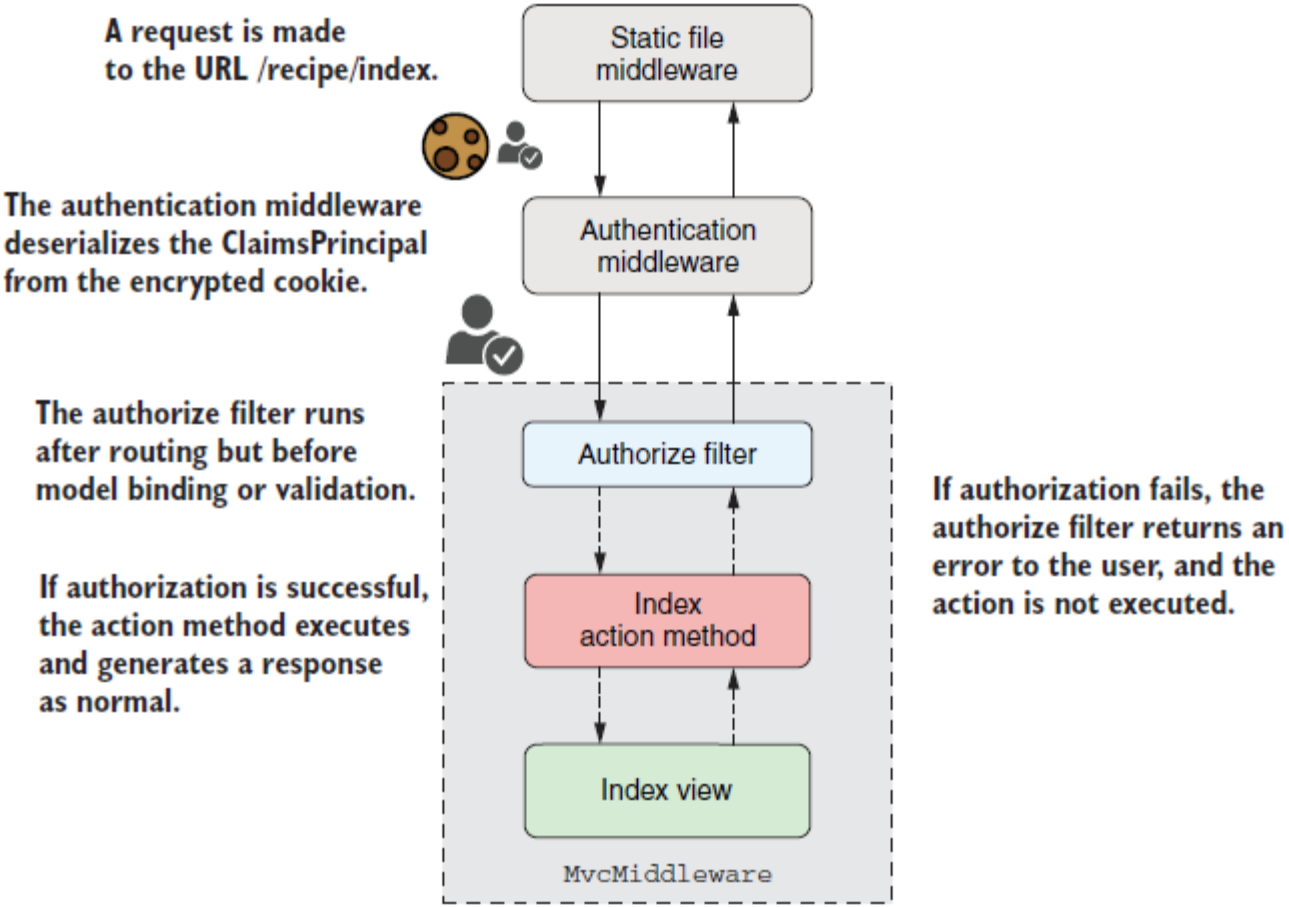
Adds the new claim to the ApplicationUser's collection

Signs the user in by setting the HttpContext.User, the principal will include the custom claim

Пример auth*

- Check-in с паспортом -> посадочный талон
- Талон -> Проверка безопасности
- Vip card -> доступ в vip зал
- Талон -> доступ в самолёт


Purchase ticket at check-in desk
Present claims: boarding pass number
Rejected
If you don't have a valid boarding pass, you're not authorized to pass through security.
At security, you must present your boarding pass number to proceed.
Pass through Security
Present claims: gold frequent flyer class
Rejected
To enter the airline lounge, you must present a Gold frequent flyer class card.
Enter airline lounge
If you're not authorized, you can't enter the airline lounge.
To board the airplane, you must present a boarding pass.
Present claims: boarding pass number
Rejected
If you're authorized, you can board the plane.
Board airplane


A request is made to the URL /recipe/index.
Static file middleware
The authentication middleware deserializes the ClaimsPrincipal from the encrypted cookie.
Authentication middleware
The authorize filter runs after routing but before model binding or validation.
Authorize filter
If authorization fails, the authorize filter returns an error to the user, and the action is not executed.
If authorization is successful, the action method executes and generates a response as normal.
Index action method
Index view
MvcMiddleware

Добавление в конвейер обработки

```
app.UseRouting();
app.UseAuthentication();
app.UseAuthorization();

app.UseEndpoints(endpoints =>
{
    endpoints.MapRazorPages();
 });
```

Сервисы настроек для Identity

```
services.Configure<IdentityOptions>(options =>{.
        options.Password.RequireDigit = true;
        options.Password.RequireLowercase = true;
        options.Password.RequireNonAlphanumeric = true;
        options.Password.RequireUppercase = true;
        options.Password.RequiredLength = 6;
        options.Password.RequiredUniqueChars = 1;
        ….
    });
```

https://docs.microsoft.com/en-us/aspnet/core/security/authentication/identity-configuration?view=aspnetcore-3.1

Сервисы настроек для Identity

```
// Lockout settings.
options.Lockout.DefaultLockoutTimeSpan = TimeSpan.FromMinutes(5);
options.Lockout.MaxFailedAccessAttempts = 5;
options.Lockout.AllowedForNewUsers = true;

// User settings.
options.User.AllowedUserNameCharacters = "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789-._@+";
options.User.RequireUniqueEmail = false;

// Default SignIn settings.
options.SignIn.RequireConfirmedEmail = false;
options.SignIn.RequireConfirmedPhoneNumber = false;
```

Настройка Cookie для Identity

```
services.ConfigureApplicationCookie(options =>
{
    options.AccessDeniedPath = "/Identity/Account/AccessDenied";
    options.Cookie.Name = "YourAppCookieName";
    options.Cookie.HttpOnly = true;
    options.ExpireTimeSpan = TimeSpan.FromMinutes(60);
    options.LoginPath = "/Identity/Account/Login";

    // ReturnUrlParameter requires
    //using Microsoft.AspNetCore.Authentication.Cookies;
    options.ReturnUrlParameter = CookieAuthenticationDefaults.ReturnUrlParameter;
    options.SlidingExpiration = true;
});
```

https://docs.microsoft.com/en-us/dotnet/api/microsoft.aspnetcore.authentication.cookies.cookieauthenticationoptions

Simple Authorization

```
[Authorize]
public class AccountController  : Controller
{ ...}

[Authorize]
public ActionResult Logout()
{ ... }

[AllowAnonymous]
public ActionResult Login()
{...}
```

Authorization Policy

- Авторизация при наличии некоторого claim’a

```
public class AirportController : Controller
{
    public IActionResult Index()
    {
        return View();
    }

    [Authorize("CanEnterSecurity")]
    public IActionResult AirportSecurity()
    {
        return View();
    }
}
```

The Index method can be executed by unauthenticated users.

Applying the "CanEnterSecurity" policy using [Authorize]

Only users that satisfy the "CanEnterSecurity" policy can execute the AirportSecurity action.

Adding authorization policy

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddAuthorization(options =>
    {
        options.AddPolicy(
            "CanEnterSecurity",
            policyBuilder => policyBuilder
                .RequireClaim("BoardingPassNumber"));
    });

    // Additional service configuration
}
```

Calls AddAuthorization to configure AuthorizationOptions

Adds a new policy

Provides a name for the policy

Defines the policy requirements using AuthorizationPolicyBuilder

Method	Policy behavior
`RequireAuthenticatedUser()`	The required user must be authenticated. Creates a policy similar to the default `[Authorize]` attribute, where you don't set a policy.
`RequireClaim(claim, values)`	The user must have the specified claim. Optionally, with one of the specified values.
`RequireUsername(username)`	The user must have the specified username.
`RequireAssertion(function)`	Executes the provided lambda function, which returns a `bool`, indicating whether the policy was satisfied.

Adding authorization policy

RequireAuthenticatedUser(): обязательная аутентификация пользователя
RequireClaim(type): для пользователя должен быть установлен claim с типом type. Причём значение неважно.
RequireClaim(type, values): для пользователя должен быть установлен claim с типом type. Но теперь claim должен в качестве значения иметь одно из значений из массива values.
RequireRole(roles): пользователь должен принадлежать к одной из ролей из массива roles
RequireUserName(name): для соответствия политике пользователь должен иметь ник (логин) name
RequireAssertion(handler): запрос должен соответствовать условию, которое устанавливается с помощью делегата handler
AddRequirements(requirement): позволяет добавить кастомное ограничение requirement, если имеющихся недостаточно

Пример

```
public class User
{
    public int Id { get; set; }
    public string Email { get; set; }
    public string Password { get; set; }
    public string City { get; set; }
    public string Company { get; set; }
    public int Year { get; set; }
}
```

```
services.AddAuthorization(opts => {
    opts.AddPolicy("OnlyForLondon", policy => {
        policy.RequireClaim(ClaimTypes.Locality, "Лондон", "London");
    });
    opts.AddPolicy("OnlyForMicrosoft", policy => {
        policy.RequireClaim("company", "Microsoft");
    });
});
```

Пример – установка Claims

```
private async Task Authenticate(User user)
    {
        // создаем один claim
        var claims = new List<Claim>
        {
            new Claim(ClaimsIdentity.DefaultNameClaimType, user.Email),
            new Claim(ClaimTypes.Locality, user.City),
            new Claim("company", user.Company)
        };
        // создаем объект ClaimsIdentity
        ClaimsIdentity id = new ClaimsIdentity(claims, "ApplicationCookie", ClaimsIdentity.DefaultNameClaimType,
            ClaimsIdentity.DefaultRoleClaimType);
        // установка аутентификационных куки
        await HttpContext.SignInAsync(CookieAuthenticationDefaults.AuthenticationScheme, new ClaimsPrincipal(id));
    }
```

Пример – авторизация

```
public class HomeController : Controller
{
    [Authorize(Policy = "OnlyForLondon")]
    public IActionResult Index()
    {
        return View();
    }

    [Authorize(Policy = "OnlyForMicrosoft")]
    public IActionResult About()
    {
        return Content("Only for Microsoft employees");
    }
}
```

Пример своего ограничения

```
using Microsoft.AspNetCore.Authorization;

public class AgeRequirement : IAuthorizationRequirement
{
    protected internal int Age { get; set; }

    public AgeRequirement(int age)
    {
        Age = age;
    }
}
```

Пример своего ограничения

```
public class AgeHandler : AuthorizationHandler<AgeRequirement>
{
    protected override Task HandleRequirementAsync(AuthorizationHandlerContext context,
        AgeRequirement requirement)
    {
        if (context.User.HasClaim(c => c.Type == ClaimTypes.DateOfBirth))
        {
            var year = 0;
            if(Int32.TryParse(context.User.FindFirst(c => c.Type == ClaimTypes.DateOfBirth).Value, out year))
            {
                if ((DateTime.Now.Year - year) >= requirement.Age)
                {
                    context.Succeed(requirement); // успешная авторизация
                }
            }
        }
        return Task.CompletedTask;
    }
}
```

Пример своего ограничения

```
// встраиваем сервис AgeHandler
services.AddTransient<IAuthorizationHandler, AgeHandler>();

services.AddAuthorization(opts => {
        // устанавливаем ограничение по возрасту
        opts.AddPolicy("AgeLimit",
            policy => policy.Requirements.Add(new AgeRequirement(18)));
    });
```

При аутентификации добавить в список claims:

```
new Claim(ClaimTypes.DateOfBirth, user.Year.ToString())
```

Imperative authorization

```
public class DocumentController : Controller
{
    private readonly IAuthorizationService _authorizationService;
    private readonly IDocumentRepository _documentRepository;

    public DocumentController(IAuthorizationService authorizationService, IDocumentRepository
documentRepository)
    {
        _authorizationService = authorizationService;
        _documentRepository = documentRepository;
    }
```

Imperative authorization

```
public async Task<IActionResult> OnGetAsync(Guid documentId)
{
    Document = _documentRepository.Find(documentId);
    if (Document == null) { return new NotFoundResult(); }

    var authorizationResult = await _authorizationService.AuthorizeAsync(User, Document, "EditPolicy");

    if (authorizationResult.Succeeded) { return Page(); }
    else
        if (User.Identity.IsAuthenticated) { return new ForbidResult(); }
        else { return new ChallengeResult(); }
}
```

Imperative authorization

```
public class DocumentAuthorizationHandler :
                AuthorizationHandler<SameAuthorRequirement, Document>
{
   protected override Task HandleRequirementAsync(AuthorizationHandlerContext context,
                                                  SameAuthorRequirement requirement,
                                                  Document resource)
   {
      if (context.User.Identity?.Name == resource.Author)
      {
                context.Succeed(requirement);
      }

      return Task.CompletedTask;
   }
}

public class SameAuthorRequirement : IAuthorizationRequirement { }
```

Imperative authorization

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

services.AddSingleton<IAuthorizationHandler, DocumentAuthorizationHandler>();
services.AddSingleton<IAuthorizationHandler, DocumentAuthorizationCrudHandler>();
services.AddScoped<IDocumentRepository, DocumentRepository>();
```

Operational requirements

```
public static class Operations
{
    public static OperationAuthorizationRequirement Create =
                new OperationAuthorizationRequirement { Name = nameof(Create) };
    public static OperationAuthorizationRequirement Read =
                new OperationAuthorizationRequirement { Name = nameof(Read) };
    public static OperationAuthorizationRequirement Update =
                new OperationAuthorizationRequirement { Name = nameof(Update) };
    public static OperationAuthorizationRequirement Delete =
                new OperationAuthorizationRequirement { Name = nameof(Delete) };
}
```

Operational requirements

```
public class DocumentAuthorizationCrudHandler :
      AuthorizationHandler<OperationAuthorizationRequirement, Document>
{
    protected override Task HandleRequirementAsync(AuthorizationHandlerContext context,
                                                   OperationAuthorizationRequirement requirement,
                                                   Document resource)
    {
            if (context.User.Identity?.Name == resource.Author &&
                    requirement.Name == Operations.Read.Name)
            {
                context.Succeed(requirement);
            }
    return Task.CompletedTask;
    }
}
```

Ссылки

https://docs.microsoft.com/en-us/aspnet/core/security/authentication/identity?view=aspnetcore-3.1&tabs=visual-studio
https://docs.microsoft.com/en-us/aspnet/core/security/authentication/social/?view=aspnetcore-3.1&tabs=visual-studio

Книга ASP.NET Core in Action - ANDREW LOCK