МІНІСТЕРСТВО ОСВІТИ ТА НАУКИ УКРАЇНИ

КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМ. ТАРАСА ШЕВЧЕНКА
ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Кафедра мережевих та інтернет технологій

СУЧАСНІ ІНТЕРНЕТ ТЕХНОЛОГІЇ

АВТЕНТИФІКАЦІЯ ТА АВТОРИЗАЦІЯ КОРИСТУВАЧІВ У ЗАСТОСУНКУ ASP.NET CORE

Лабораторне заняття № 4

Заяць Діани Юріївни

Хід виконання роботи:

Завдання 1. Забезпечте автентифікацію користувачів у застосунку. Реалізуйте сторінку входу, реєстрації та виходу, використовуючи ASP.NET Core Identity. Перевірте, що неавтентифіковані користувачі не мають доступу до жодної сторінки, крім головної сторінки та сторінки реєстрації/автентифікації.

Для виконання першого завдання у файлі Program.cs було налаштовано глобальну політику авторизації для MVC-контролерів, яка за замовчуванням вимагає автентифікації від усіх користувачів. Тобто кожний запит до контролерів MVC обробляється лише після успішного входу в систему.

```
85
86      // MVC controllers: by default require authenticated user
87      builder.Services.AddControllersWithViews(options =>
88      {
89          var policy = new AuthorizationPolicyBuilder()
90              .RequireAuthenticatedUser()
91              .Build();
92          options.Filters.Add(new AuthorizeFilter(policy));
93      });
94      builder.Services.AddRazorPages();
95
```

Рисунок 4.1 - Налаштування глобальної політики авторизації для контролерів MVC (RequireAuthenticatedUser)

Додатково були підключені необхідні простори імен (using), що забезпечують роботу механізмів авторизації та фільтрів MVC, які застосовуються для контролю доступу

```
11      using Microsoft.AspNetCore.Authorization;
12      using Microsoft.AspNetCore.Mvc.Authorization;
13      using System.Runtime.InteropServices;
14      using WebAppCore.Authorization;
15
```

Рисунок 4.2 - Директиви using для підключення класів авторизації та MVC-фільтрів

Для дії Index у HomeController було встановлено атрибут AllowAnonymous, що дає змогу неавтентифікованим користувачам переглядати головну сторінку, навіть попри глобальну вимогу автентифікації. Таким чином, головна сторінка залишається загальнодоступною.

Рис 4.3 - Використання [AllowAnonymous] у методі Index контролера

Сторінки в Areas/Identity/Pages/Account (зокрема Login та Register) зазвичай уже мають атрибут AllowAnonymous у типовій конфігурації ASP.NET Core Identity. Завдяки цьому сторінки входу та реєстрації залишаються доступними для неавтентифікованих користувачів, навіть коли більшість інших сторінок захищені. У такому режимі користувач, який не виконав вхід, може переглядати лише головну сторінку (Home/Index) та сторінки логіну/реєстрації.

Для моделі сторінки входу (Login.cshtml.cs) атрибут AllowAnonymous був явно доданий до класу моделі, що гарантує доступ до цієї сторінки без попередньої автентифікації, навіть у разі зміни глобальних налаштувань у майбутньому.

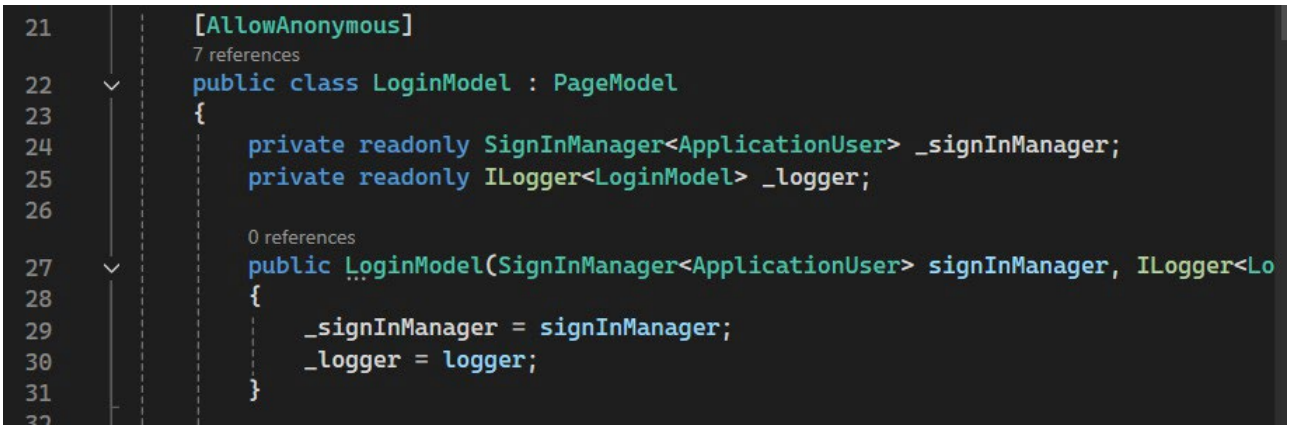


Рисунок 4.4 – AllowAnonymous для моделі сторінки логіну

Аналогічний підхід було застосовано до Register Model:

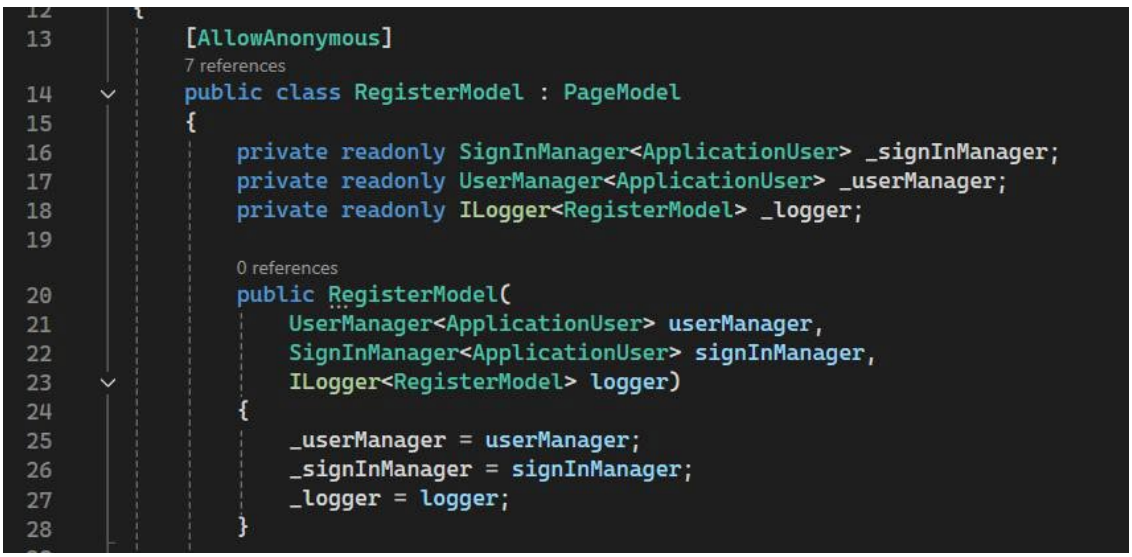


Рисунок 4.5 - AllowAnonymous для моделі сторінки реєстрації

Для всіх інших сторінок MVC-застосунку налаштовано перенаправлення неавтентифікованих користувачів на сторінку входу. У разі спроби доступу до захищеного ресурсу без автентифікації користувач автоматично перенаправляється на сторінку логіну, а у випадку відмови в авторизації – на сторінку AccessDenied.

```
55      // Configure application cookie (login path, etc.)
56      builder.Services.ConfigureApplicationCookie(options =>
57      {
58          options.LoginPath = "/Identity/Account/Login";
59          options.AccessDeniedPath = "/Identity/Account/AccessDenied";
60      });
```

Рисунок 4.6 - Встановлення маршрутів для перенаправлення на логін та сторінку AccessDenied

Завдання 2. Створіть політику авторизації, яка дозволяє доступ до сторінки «Архів матеріалів» лише тим користувачам, які мають твердження IsVerifiedClient. Додайте твердження вручну під час реєстрації.

Для реалізації другої вимоги у конфігурації авторизації було оголошено окрему політику, що перевіряє наявність у користувача claim'у IsVerifiedClient. Лише користувачі, для яких це твердження встановлено, можуть отримати доступ до сторінки «Архів матеріалів».

```
// Authorization policies
builder.Services.AddAuthorization(options =>
{
    // Policy for users that have IsVerifiedClient claim
    options.AddPolicy("VerifiedClientOnly", policy =>
        policy.RequireClaim("IsVerifiedClient", "true"));
```

Рисунок 4.7 – Створення політики авторизаціїї для веріфікованих клієнтів

На етапі реєстрації нового користувача (у файлі Register.cshtml.cs) після успішного створення облікового запису до нього програмно додається claim IsVerifiedClient=true. Це означає, що щойно зареєстрований користувач одразу позначається як «верифікований клієнт» і може користуватися функціональністю, яка захищена відповідною політикою (зокрема, переглядати «Архів матеріалів»).

```
        await _userManager.AddClaimAsync(user,
            new System.Security.Claims.Claim("IsVerifiedClient", "true"));

        // Додаємо WorkingHours claim для доступу до Premium сторінки
        // Значення 150 дозволяє доступ (мінімум 100)
        await _userManager.AddClaimAsync(user,
            new System.Security.Claims.Claim("WorkingHours", "150"));

        await _signInManager.SignInAsync(user, isPersistent: false);
        return LocalRedirect(returnUrl);
    }
```

Рисунок 4.8 – Додавання claim при реєстрації

Завдання 3. Реалізуйте ресурсну авторизацію для сторінки редагування ресурсу. Кожен ресурс має автора, і лише автор може редагувати його. Використайте `IAuthorizationService` та створіть обробник, який перевіряє, чи поточний користувач є автором ресурсу.

Для реалізації ресурсної авторизації було створено окрему сутність ресурсу Post, яка містить ідентифікатор автора. До моделі було додано властивість AuthorId, що зберігає ідентифікатор користувача системи (User.Id), а також, за потреби, навігаційну властивість до сутності користувача. Це дозволяє однозначно встановити, хто саме є власником кожного ресурсу.

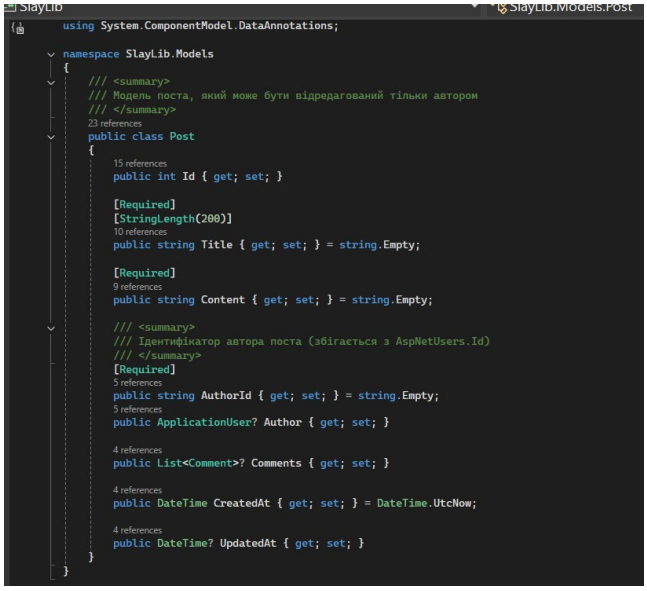


Рисунок 4.9 – Модель ресурсу Post з полем AuthorId для збереження автора

Під час створення нового ресурсу в дії Create контролера значення AuthorId встановлюється програмно на основі поточного автентифікованого користувача, що отримується через UserManager або з об’єкта User. При цьому поле AuthorId не відображається у формі та не приймається від користувача, а також виключається з ModelState, щоб уникнути помилок валідації при збереженні. Таким чином, користувач не може змінити автора ресурсу вручну.

Для підтримки ресурсної авторизації було створено вимогу IsResourceOwnerRequirement, яка реалізує інтерфейс IAuthorizationRequirement. Ця вимога не містить додаткової логіки, а виступає маркером, що позначає необхідність перевірки права власності на ресурс.

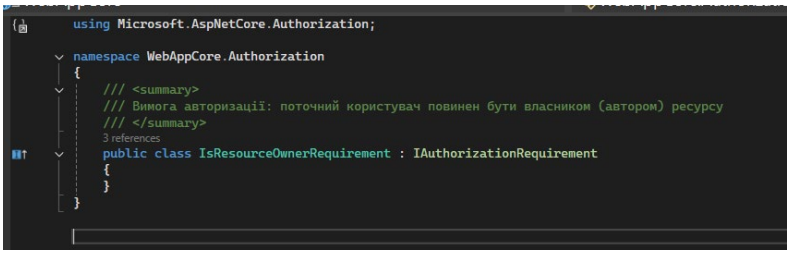


Рисунок 4.10 – Оголошення вимоги авторизації IsResourceOwnerRequirement

Далі було реалізовано обробник IsResourceOwnerHandler, що наслідується від

AuthorizationHandler<IsResourceOwnerRequirement, Post>. У методі HandleRequirementAsync обробник отримує поточного користувача з контексту (context.User), витягує його ідентифікатор та порівнює його зі значенням AuthorId у переданому ресурсі Post. Якщо ідентифікатори збігаються, викликається context.Succeed(requirement), що означає успішну авторизацію. У протилежному випадку вимога не задовольняється, і доступ до редагування блокується.

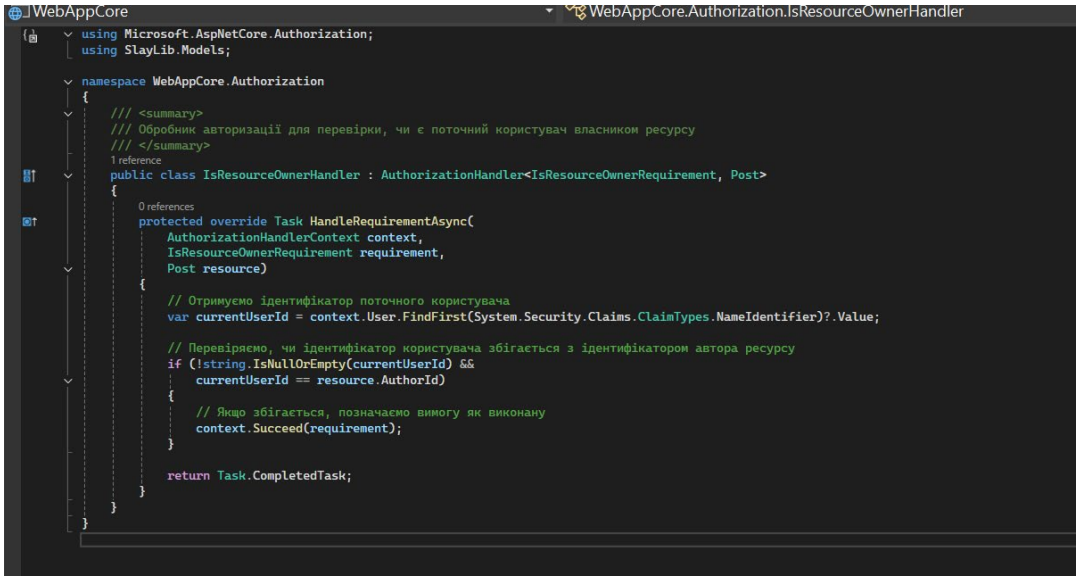


Рисунок 4.11 – Реалізація обробника IsResourceOwnerHandler з порівнянням AuthorId та ідентифікатора поточного користувача

Обробник було зареєстровано в контейнері залежностей за допомогою виклику services.AddScoped<IAuthorizationHandler, IsResourceOwnerHandler>() у файлі Program.cs. Паралельно була створена політика, наприклад "CanEditResource", до якої додається вимога IsResourceOwnerRequirement. Після цього контролер PostController, що відповідає за роботу з ресурсами, отримує через конструктор інтерфейс IAuthorizationService.

У діях Edit (GET і POST) ресурс спочатку завантажується з бази даних, після чого викликається перевірка:

var authResult = await _authorizationService.AuthorizeAsync(User, post, "CanEditResource");

Якщо authResult.Succeeded == false, контролер повертає результат Forbid() або Challenge(), що призводить до відмови в доступі (403) або перенаправлення на сторінку входу. Таким чином, навіть автентифікований користувач не може редагувати ресурс, який йому не належить.

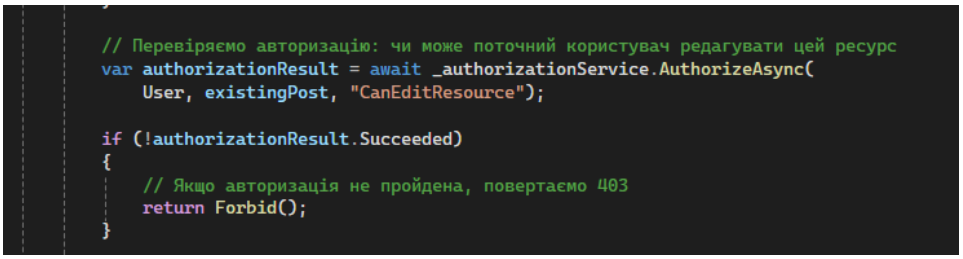


Рисунок 4.12 – Використання IAuthorizationService у методі редагування ресурсу для перевірки права власності

Завдання 4. Створіть кастомну вимогу авторизації MinimumWorkingHoursRequirement, яка дозволяє доступ до сторінки «Преміум» лише тим користувачам, які мають твердження WorkingHours з числовим значенням не менше 100. Реалізуйте обробник, який виконує перевірку

Для обмеження доступу до сторінки «Преміум» за числовим значенням твердження було створено окрему вимогу MinimumWorkingHoursRequirement, що реалізує IAuthorizationRequirement і містить властивість MinimumHours. У рамках даної лабораторної роботи мінімальне значення встановлено рівним 100. Це дозволяє гнучко змінювати порогове значення без модифікації логіки перевірки.

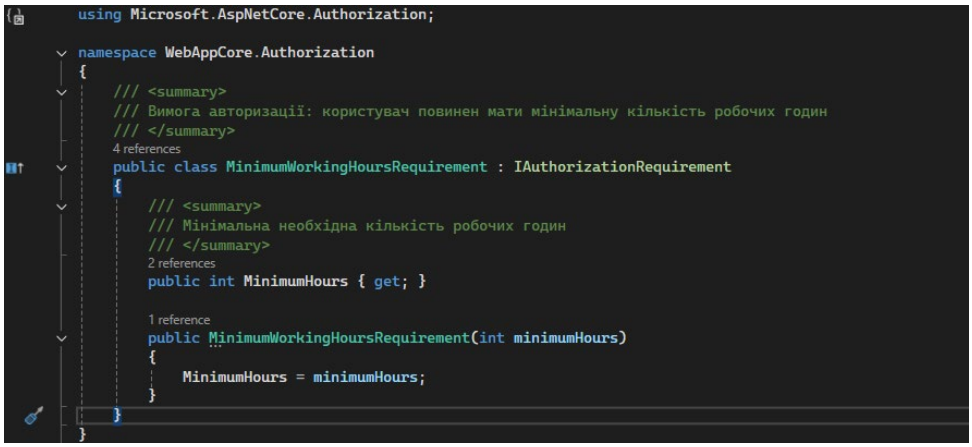


Рисунок 4.13 – Клас вимоги MinimumWorkingHoursRequirement з параметром мінімальної кількості годин

Обробка цієї вимоги виконується в класі MinimumWorkingHoursHandler, який наслідується від AuthorizationHandler<MinimumWorkingHoursRequirement>. У методі HandleRequirementAsync обробник отримує з об’єкта User claim з назвою WorkingHours, намагається перетворити його значення в тип int та порівнює отриманий результат з requirement.MinimumHours. Якщо користувач має твердження WorkingHours і його значення не менше 100, викликається context.Succeed(requirement), що надає доступ до сторінки «Преміум». У всіх інших випадках вимога вважається незадоволеною, і доступ блокується.

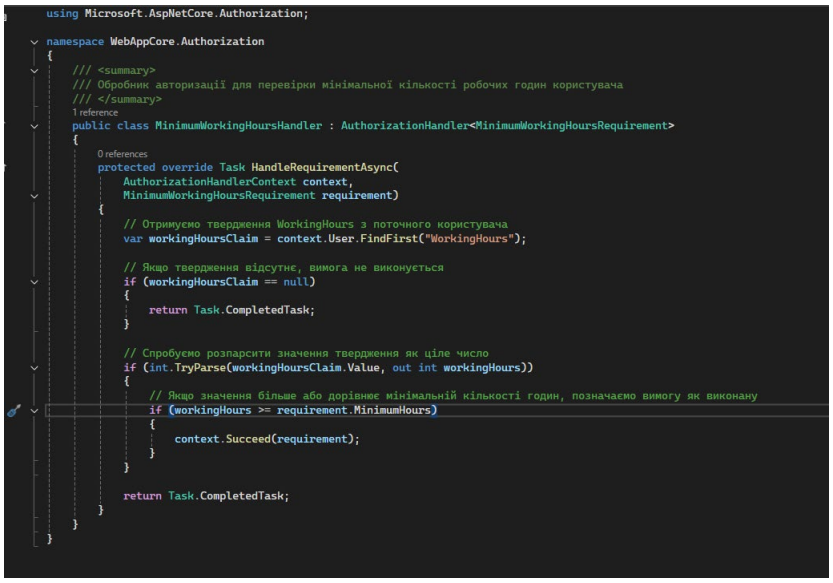


Рисунок 4.14 – Обробник MinimumWorkingHoursHandler з перевіркою числового значення WorkingHours

У Program.cs обробник було зареєстровано через services.AddScoped<IAuthorizationHandler, MinimumWorkingHoursHandler>(). Також була додана політика авторизації, наприклад "CanAccessPremium", у якій створюється екземпляр MinimumWorkingHoursRequirement із параметром 100.

На дію контролера, що відповідає за відображення сторінки «Преміум» Premium.Index, було додано атрибут [Authorize(Policy = "CanAccessPremium")]. Це забезпечує автоматичну перевірку вимоги перед виконанням дії. Якщо користувач не має відповідного claim або його значення менше 100, фреймворк повертає помилку доступу (403) або перенаправляє на сторінку AccessDenied.

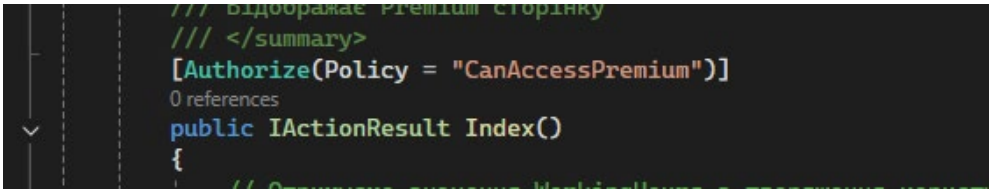
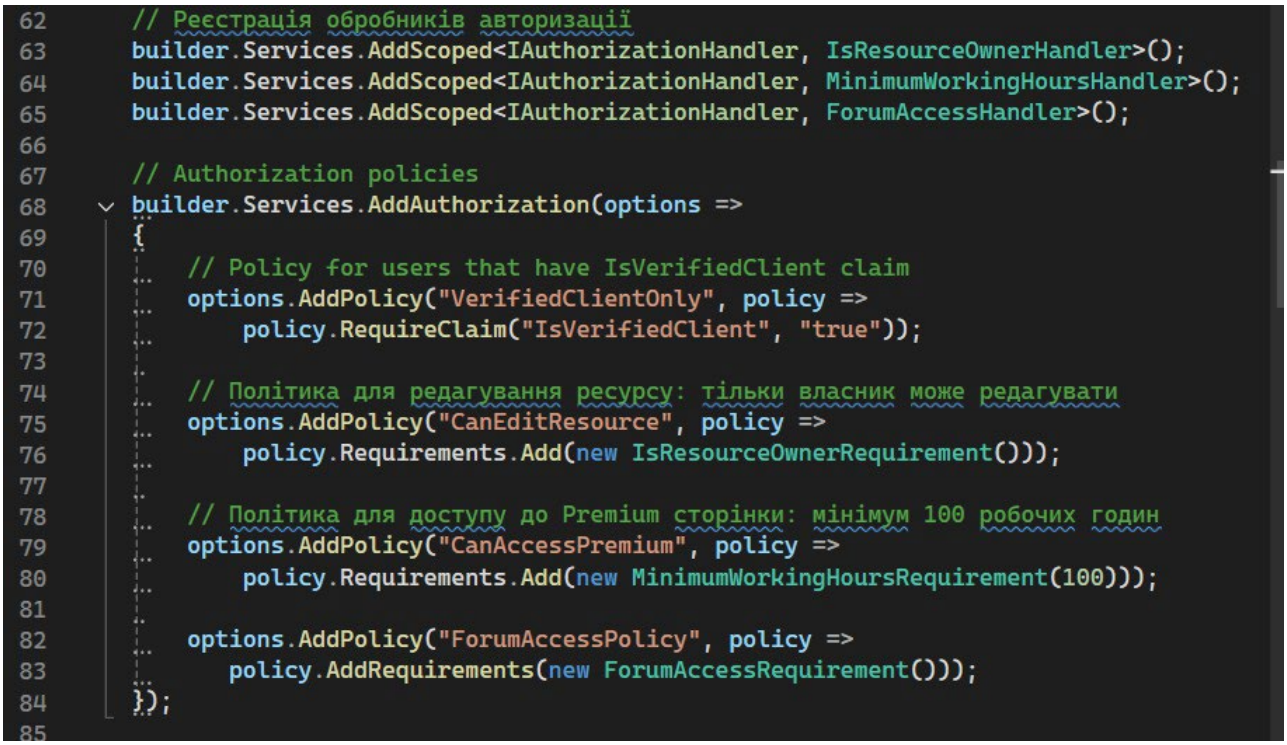


Рисунок 4.15 – Застосування політики CanAccessPremium до дії контролера сторінки «Преміум»

Для цілей тестування може використовуватися допоміжна сторінка або процедура, що дозволяє створюваним користувачам автоматично додавати claim WorkingHours (наприклад, зі значенням 150) під час реєстрації. Це спрощує демонстрацію роботи політики й дозволяє перевіряти як позитивні, так і негативні сценарії, змінюючи значення WorkingHours нижче порогу (наприклад, 50).

Завдання 5. Створіть політику, яка дозволяє доступ до сторінки «Форум» лише тим користувачам, які мають хоча б одне з тверджень: IsMentor, IsVerifiedUser, або HasForumAccess. Реалізуйте обробник, який перевіряє хоча б одне з цих тверджень.



```
// Реєстрація обробників авторизації
builder.Services.AddScoped<IAuthorizationHandler, IsResourceOwnerHandler>();
builder.Services.AddScoped<IAuthorizationHandler, MinimumWorkingHoursHandler>();
builder.Services.AddScoped<IAuthorizationHandler, ForumAccessHandler>();

// Authorization policies
builder.Services.AddAuthorization(options =>
{
    // Policy for users that have IsVerifiedClient claim
    options.AddPolicy("VerifiedClientOnly", policy =>
        policy.RequireClaim("IsVerifiedClient", "true"));

    // Політика для редагування ресурсу: тільки власник може редагувати
    options.AddPolicy("CanEditResource", policy =>
        policy.Requirements.Add(new IsResourceOwnerRequirement()));

    // Політика для доступу до Premium сторінки: мінімум 100 робочих годин
    options.AddPolicy("CanAccessPremium", policy =>
        policy.Requirements.Add(new MinimumWorkingHoursRequirement(100)));

    options.AddPolicy("ForumAccessPolicy", policy =>
       policy.AddRequirements(new ForumAccessRequirement()));
});
```

4.9 – Реєстрації обробників авторизації та створення політик доступ

Висновок:

У межах лабораторної роботи було розглянуто та реалізовано на практиці основні механізми безпеки в ASP.NET Core. Зокрема, досліджено автентифікацію як процес встановлення особи користувача та авторизацію як механізм визначення дозволених для нього дій. Для керування обліковими записами та їхніми твердженнями (claims) було використано ASP.NET Core Identity. Окрему увагу приділено ресурсній авторизації із залученням IAuthorizationService та власних обробників, що дає змогу здійснювати тонке налаштування доступу до конкретних ресурсів. У результаті було сформовано практичні навички побудови надійної та гнучкої системи розмежування прав доступу у вебзастосунку.