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

Черкун Єви Сергіївни

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.

У Program.cs було додано політику, щоб усі MVC-контролери за замовчуванням вимагали автентифікацію:

```
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)

Також були додані using-и:

```
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 HomeController.Index додано AllowAnonymous, щоб неавтентифіковані користувачі могли бачити тільки головну.

```
[AllowAnonymous]
0 references
public async Task<IActionResult> Index()
{
    var users = await _mitRepository.GetAllAsync<ApplicationUser>();
    return View(users);
}

0 references
public IActionResult Privacy()
{
    return View();
}
```

Рис 4.3 - Використання [AllowAnonymous] у методі Index контролера

Сторінки в Areas/Identity/Pages/Account зазвичай уже додамо AllowAnonymous (Login, Register), тому вони й далі будуть доступні для неавтентифікованих користувачів. Неавтентифікований користувач бачить тільки головну (Home/Index) та сторінки логіну/реєстрації Identity

У Login.cshtml.cs додано атрибут AllowAnonymous для моделі сторінки логіну

```
21      [AllowAnonymous]
        7 references
22      public class LoginModel : PageModel
23      {
24          private readonly SignInManager<ApplicationUser> _signInManager;
25          private readonly ILogger<LoginModel> _logger;
26
            0 references
27          public LoginModel(SignInManager<ApplicationUser> signInManager, ILogger<Lo
28          {
29              _signInManager = signInManager;
30              _logger = logger;
31          }
```

Рисунок 4.4 – AllowAnonymous для моделі сторінки логіну

Так само робимо з Register Model:

```
13      [AllowAnonymous]
        7 references
14      public class RegisterModel : PageModel
15      {
16          private readonly SignInManager<ApplicationUser> _signInManager;
17          private readonly UserManager<ApplicationUser> _userManager;
18          private readonly ILogger<RegisterModel> _logger;
19
            0 references
20          public RegisterModel(
21              UserManager<ApplicationUser> userManager,
22              SignInManager<ApplicationUser> signInManager,
23              ILogger<RegisterModel> logger)
24          {
25              _userManager = userManager;
26              _signInManager = signInManager;
27              _logger = logger;
28          }
```

Рисунок 4.5 - AllowAnonymous для моделі сторінки реєстрації

Будь-яка інша сторінка MVC буде вимагати вхід у систему (редірект на логін):

```
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.

Додаємо політику авторизації, що мають підтвердження IsVerifiedClient

```
// 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

У Register.cshtml.cs після успішного створення користувача додається твердження IsVerifiedClient=true:

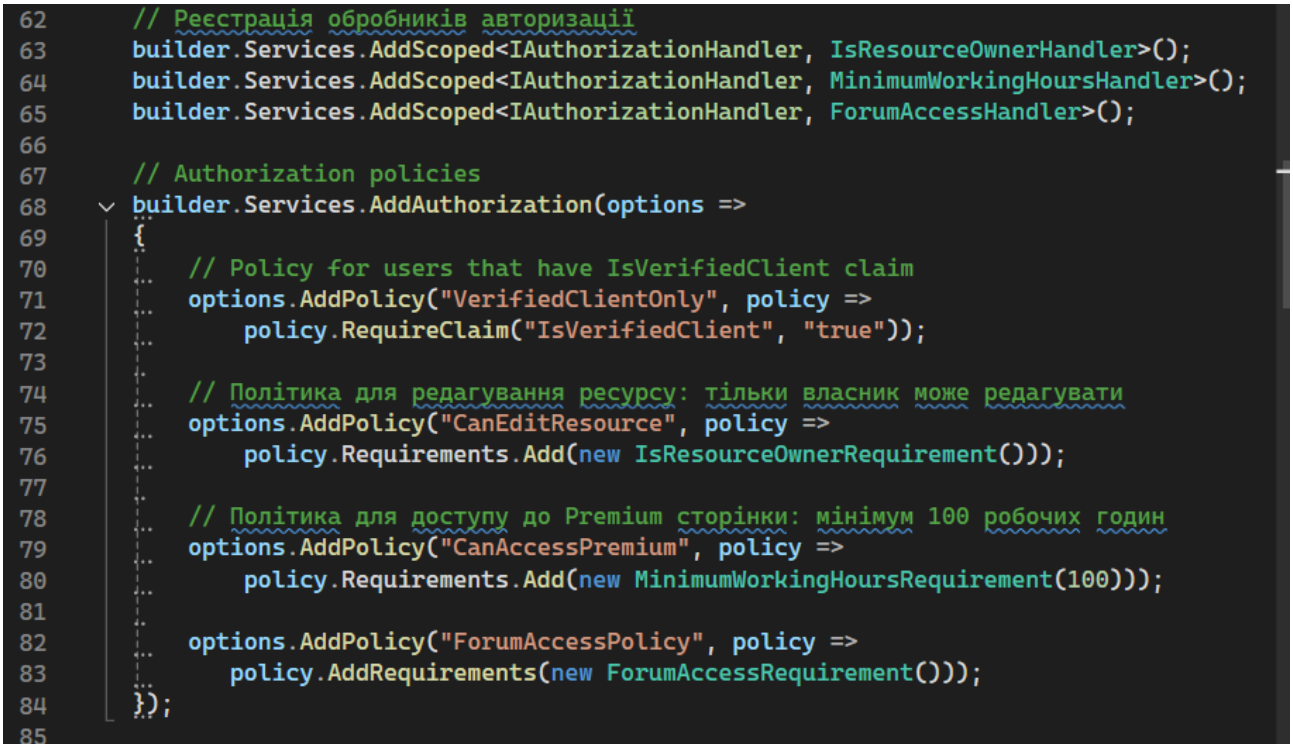
```
        await _userManager.AddClaimAsync(user,
            new System.Security.Claims.Claim("IsVerifiedClient", "true"));

        // 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.

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

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:

В ході лабораторного заняття було вивчено та практично реалізовано ключові механізми безпеки в ASP.NET Core: автентифікацію — процес встановлення особи користувача, який передує авторизації — процесу визначення дозволених дій. Було застосовано ASP.NET Core Identity для управління користувачами та їхніми твердженнями (claims). Опановано складну ресурсну авторизацію з використанням IAuthorizationService та кастомних обробників, що дозволяє обмежувати доступ до конкретних ресурсів. Таким чином, набуто навичок створення надійної та гнучкої системи розмежування доступу в вебзастосунку.