ЛАБОРАТОРНА РОБОТА № 5

Аутентифікація та авторизація. Робота з Web API. JSON Web Token.

Мета: набути навичок роботи з механізмами аутентифікації та авторизації користувачів в ASP.NET, набути навичок роботи з Web API.

Хід роботи:

Завдання 1

Реалізувати реєстрацію, автентифікацію та авторизацію користувача з використанням пакету Microsoft Identity.

Листинг Program.cs після виконання лабораторної роботи:

```
using Microsoft.AspNetCore.Identity;
using Microsoft.EntityFrameworkCore;
using SurveyPortal.Models.Identity.Entities;
using SurveyPortal.Models.Identity;
using SurveyPortal.Models.Survey.Survey;
using SurveyPortal.Models;
using SurveyPortal.Services;

var builder = WebApplication.CreateBuilder(args);


builder.Services.AddControllersWithViews();
builder.Services.AddDistributedMemoryCache();

builder.Services.AddSession(options =>
{
    options.IdleTimeout = TimeSpan.FromSeconds(10);
    options.Cookie.HttpOnly = true;
    options.Cookie.IsEssential = true;
});

builder.Services.AddDbContext<SurveyDbContext>(opts =>
{
    var connectionString = builder.Configuration.GetConnectionString("SurveyPortalConnection");
    opts.UseMySql(connectionString, ServerVersion.AutoDetect(connectionString));
});

builder.Services.AddDbContext<AppIdentityDbContext>(opts =>
{
    var connectionString = builder.Configuration.GetConnectionString("IdentityConnection");
    opts.UseMySql(connectionString, ServerVersion.AutoDetect(connectionString));
});

builder.Services.AddIdentity<User, IdentityRole>(options =>
{
    options.Password.RequiredLength = 8;
    options.Password.RequireDigit = true;
    options.Password.RequireLowercase = true;
    options.Password.RequireUppercase = true;
    options.Password.RequireNonAlphanumeric = false;
    options.User.RequireUniqueEmail = true;
})
.AddEntityFrameworkStores<AppIdentityDbContext>()
.AddDefaultTokenProviders();

builder.Services.ConfigureApplicationCookie(options =>
{
    options.LoginPath = "/Accounts/Login";
    options.AccessDeniedPath = "/Accounts/AccessDenied";
});

builder.Services.AddScoped<ISurveyRepository, EFSurveyRepository>();
builder.Services.AddScoped<IFileService, FileService>();

builder.Services.AddAutoMapper(typeof(Program));
builder.Services.AddAutoMapper(typeof(MappingProfile));
```

```
builder.WebHost.ConfigureKestrel(options =>
{
    options.Limits.MaxRequestBodySize = 10 * 1024 * 1024;
});

var app = builder.Build();

app.UseStaticFiles();
app.UseSession();

app.MapDefaultControllerRoute();

app.MapControllerRoute(
    name: "pagination",
    pattern: "Home/Index/{page:int}",
    defaults: new { Controller = "Home", action = "Index" }
);

SeedData.EnsurePopulated(app);

var scope = app.Services.CreateScope();
var roleManager = scope.ServiceProvider.GetRequiredService<RoleManager<IdentityRole>>();
string[] roles = { "Visitor", "Admin" };

foreach (var role in roles)
{
    if (!await roleManager.RoleExistsAsync(role))
    {
        await roleManager.CreateAsync(new IdentityRole(role));
    }
}

app.Run();
```

Листинг AppIdentityDbContext після виконання лабораторної роботи:

```
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Identity.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore;
using SurveyPortal.Models.Identity.Entities;
namespace SurveyPortal.Models.Identity
{
    public class AppIdentityDbContext : IdentityDbContext<User>
    {
        public AppIdentityDbContext(
        DbContextOptions<AppIdentityDbContext> options)
        : base(options) { }

        protected override void OnModelCreating(ModelBuilder modelBuilder)
        {
            base.OnModelCreating(modelBuilder);
        }
    }
}
```

Листинг RoleManagerService.cs після виконання лабораторної роботи:

```
using Microsoft.AspNetCore.Identity;
using SurveyPortal.Models.Identity.Entities;
using System.Threading.Tasks;

namespace SurveyPortal.Services
{
    public class RoleManagerService
    {
        private readonly UserManager<User> _userManager;
        private readonly RoleManager<IdentityRole> _roleManager;

        public RoleManagerService(UserManager<User> userManager, RoleManager<IdentityRole> roleManager)
        {
            _userManager = userManager;
            _roleManager = roleManager;
        }

        public async Task<bool> AddUserToAdminRoleAsync(string userIdOrEmail)
        {
            User user = await _userManager.FindByIdAsync(userIdOrEmail);

            if (user == null)
            {
                user = await _userManager.FindByEmailAsync(userIdOrEmail);
            }

            if (user == null)
            {
            }

            if (!await _roleManager.RoleExistsAsync("Admin"))
            {
                await _roleManager.CreateAsync(new IdentityRole("Admin"));
            }

            if (await _userManager.IsInRoleAsync(user, "Admin"))
            {
                return true;
```

```
            }

            var result = await _userManager.AddToRoleAsync(user, "Admin");
            return result.Succeeded;
        }

        public async Task<bool> RemoveUserFromRoleAsync(string userIdOrEmail, string roleName)
        {
            User user = await _userManager.FindByIdAsync(userIdOrEmail);

            if (user == null)
            {
                user = await _userManager.FindByEmailAsync(userIdOrEmail);
            }

            if (user == null)
            {
                return false;
            }

            if (!await _userManager.IsInRoleAsync(user, roleName))
            {
                return true;
            }

            var result = await _userManager.RemoveFromRoleAsync(user, roleName);
            return result.Succeeded;
        }
    }
     }
```

Листинг appsettings.json після виконання лабораторної роботи:

```
{
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft.AspNetCore": "Warning"
    }
  },
  "AllowedHosts": "*",
  "ConnectionStrings": {
    "SurveyPortalConnection": "server=localhost;database=surveyportal;user=homeuser;password=",
    "IdentityConnection": "server=localhost;database=identity;user=homeuser;password="
  }
}
```

Після конфігурації Identity, реалізуйте наступні функціональності:

Реєстрація користувача

встановити вимогу по унікальності електронної адреси користувача

встановити вимоги до пароля: не коротший за 8 символів, містить цифри та літери, обов’язково містить хоча б одну літеру в верхньому регістрі

реалізувати поле для підтвердження введеного паролю

Автентифікація користувача (Login)

Вихід користувача (Logout)

Особистий кабінет користувача, де він матиме змогу переглядати або змінювати дані про себе.

Розробіть всі необхідні форми з підтримкою серверної валідації та розробіть відповідний контроллер. Налаштуйте доступ до Action-методів контроллерів з використанням атрибуту [Authorize] – лише залогінений користувач повинен мати можливості виконувати дії на веб-сайті, для неавтентифікованого користувача має бути доступною лише реєстрація та логін.

Листинг AccountsController.cs після виконання лабораторної роботи:

```
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Mvc;
using SurveyPortal.Models.Identity.Entities;
using AutoMapper;
using SurveyPortal.Models.Identity.DTO;
using Microsoft.AspNetCore.Authorization;

namespace SurveyPortal.Controllers
{
    public class AccountsController : Controller
    {
        private readonly SignInManager<User> _signInManager;
        private readonly UserManager<User> _userManager;
        private readonly IMapper _mapper;
```

```
        public AccountsController(SignInManager<User> signInManager, UserManager<User> userManager, IMapper mapper)
        {
            _signInManager = signInManager;
            _mapper = mapper;
            _userManager = userManager;
        }

        public IActionResult Login()
        {
            return View();
        }

        [HttpPost]
        public async Task<IActionResult> Login(UserForLoginDto userModel)
        {
            if (ModelState.IsValid)
            {
                var result = await _signInManager.PasswordSignInAsync(userModel.Email, userModel.Password, userModel.RememberMe,
false);
                if (result.Succeeded)
                {
                    return RedirectToAction("Index", "Home");
                }
                else
                {
                    ModelState.AddModelError(string.Empty, "Email or password is incorrect.");
                    return View(userModel);
                }

            }
            return View(userModel);
        }

        public IActionResult VerifyEmail()
        {
            return View();
        }

        [HttpPost]
        public async Task<IActionResult> VerifyEmail(EmailVerificationDto userModel)
        {
            if (ModelState.IsValid)
            {
                var user = await _userManager.FindByEmailAsync(userModel.Email);
                if (user == null)
                {
                    ModelState.AddModelError("Email", "No account found with this email address.");
                    return View(userModel);
                }

                else
                    return RedirectToAction("ChangePassword", "Accounts", new {username = userModel.Email});
            }

            return View(userModel);
        }



        public IActionResult ChangePassword(string userName)
        {
            if (string.IsNullOrEmpty(userName))
            {
                return RedirectToAction("VerifyEmail", "Accounts");
            }
            return View(new ChangePasswordDto { Email = userName });
        }

        [HttpPost]
        public async Task<IActionResult> ChangePassword(ChangePasswordDto userModel)
        {
            if (ModelState.IsValid)
            {
                var user = await _userManager.FindByNameAsync(userModel.Email);
                if (user != null)
                {
                    var result = await _userManager.RemovePasswordAsync(user);
                    if (result.Succeeded)
                    {
                        result = await _userManager.AddPasswordAsync(user, userModel.NewPassword);
                        if (result.Succeeded)
                        {
                            return RedirectToAction("Login", "Accounts");
                        }
                        else
                        {
                            ModelState.AddModelError("", "Failed to set new password.");
                        }
                    }
                    else
                    {
                        ModelState.AddModelError("", "Failed to remove old password.");
                    }
                }
                else
```

```
            {
                ModelState.AddModelError("", "Email not found!");
            }
        }
        else
        {
            ModelState.AddModelError("", "Invalid input data.");
        }

        return View(userModel);
    }

    [HttpGet]
    [HttpGet]
    public IActionResult Register()
    {
        return View();
    }

    [HttpPost]
    [ValidateAntiForgeryToken]
    public async Task<IActionResult> Register(UserForRegistrationDto userModel)
    {
        if (!ModelState.IsValid)
        {
            return View(userModel);
        }

        var existingUser = await _userManager.FindByEmailAsync(userModel.Email);
        if (existingUser != null)
        {
            ModelState.AddModelError("Email", "Email is already taken.");
            return View(userModel);
        }
        var user = _mapper.Map<User>(userModel);

        user.UserName = userModel.Email;

        var result = await _userManager.CreateAsync(user, userModel.Password);

        if (!result.Succeeded)
        {
            return View(userModel);
        }

        await _userManager.AddToRoleAsync(user, "Visitor");

        return RedirectToAction("Login", "Accounts");
    }


    [Authorize]
    public async Task<IActionResult> Logout()
    {
        await _signInManager.SignOutAsync();
        return RedirectToAction("Index", "Home");
    }


    [Authorize]
    [HttpGet]
    public async Task<IActionResult> Profile()
    {
        var user = await _userManager.GetUserAsync(User);
        if (user == null)
        {
            return RedirectToAction("Login");
        }

        var userProfileDto = _mapper.Map<UserProfileDto>(user);
        return View(userProfileDto);
    }


    [Authorize]
    [HttpPost]
    [ValidateAntiForgeryToken]
    public async Task<IActionResult> Profile(UserProfileDto model, IFormFile profilePicture)
    {
        if (!ModelState.IsValid)
        {
            return View(model);
        }

        var user = await _userManager.GetUserAsync(User);
        if (user == null)
        {
            return RedirectToAction("Login");
        }

        user.FirstName = model.FirstName;
        user.LastName = model.LastName;
        user.PhoneNumber = model.PhoneNumber;
        user.DateOfBirth = model.DateOfBirth;
        user.Address = model.Address;
```

```
            user.City = model.City;
            user.State = model.State;
            user.PostalCode = model.PostalCode;
            user.Country = model.Country;

            if (user.Email != model.Email)
            {
                var setEmailResult = await _userManager.SetEmailAsync(user, model.Email);
                if (!setEmailResult.Succeeded)
                {
                    foreach (var error in setEmailResult.Errors)
                    {
                        ModelState.AddModelError(string.Empty, error.Description);
                    }
                    return View(model);
                }
                var setUsernameResult = await _userManager.SetUserNameAsync(user, model.Email);
                if (!setUsernameResult.Succeeded)
                {
                    foreach (var error in setUsernameResult.Errors)
                    {
                        ModelState.AddModelError(string.Empty, error.Description);
                    }
                    return View(model);
                }
            }

            if (profilePicture != null && profilePicture.Length > 0)
            {
                try
                {
                    var allowedExtensions = new[] { ".jpg", ".jpeg", ".png", ".gif" };
                    var fileExtension = Path.GetExtension(profilePicture.FileName).ToLowerInvariant();

                    if (!allowedExtensions.Contains(fileExtension))
                    {
                        ModelState.AddModelError("ProfilePicture", "Only image files (jpg, jpeg, png, gif) are allowed.");
                        return View(model);
                    }

                    if (profilePicture.Length > 5 * 1024 * 1024)
                    {
                        ModelState.AddModelError("ProfilePicture", "File size cannot exceed 5MB.");
                        return View(model);
                    }

                    var uploadsDir = Path.Combine(Directory.GetCurrentDirectory(), "wwwroot", "uploads", "profiles");
                    if (!Directory.Exists(uploadsDir))
                    {
                        Directory.CreateDirectory(uploadsDir);
                    }

                    var fileName = $"{user.Id}_{Guid.NewGuid()}{fileExtension}";
                    var filePath = Path.Combine(uploadsDir, fileName);

                    using (var stream = new FileStream(filePath, FileMode.Create))
                    {
                        await profilePicture.CopyToAsync(stream);
                    }

                    user.ProfilePictureUrl = $"/uploads/profiles/{fileName}";
                }
                catch (Exception ex)
                {
                    ModelState.AddModelError("", $"Error uploading file: {ex.Message}");
                    return View(model);
                }
            }

            var result = await _userManager.UpdateAsync(user);
            if (result.Succeeded)
            {
                TempData["SuccessMessage"] = "Your profile has been updated successfully!";
                return RedirectToAction("Profile");
            }

            foreach (var error in result.Errors)
            {
                ModelState.AddModelError(string.Empty, error.Description);
            }

            return View(model);
        }

        public IActionResult AccessDenied()
        {
            return View();
        }
    }
}
```

Листинг AdminController.cs після виконання лабораторної роботи:

```
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Mvc;
```

```
using Microsoft.EntityFrameworkCore;
using SurveyPortal.Models.Identity;
using SurveyPortal.Models.Identity.Entities;
using System.Threading.Tasks;

namespace SurveyPortal.Controllers
{
    public class AdminController : Controller
    {
        private readonly UserManager<User> _userManager;
        private readonly RoleManager<IdentityRole> _roleManager;
        private readonly AppIdentityDbContext _dbContext;

        public AdminController(
            UserManager<User> userManager,
            RoleManager<IdentityRole> roleManager,
            AppIdentityDbContext dbContext)
        {
            _userManager = userManager;
            _roleManager = roleManager;
            _dbContext = dbContext;
        }

        public async Task<IActionResult> UserManagement()
        {
            var users = await _userManager.Users.ToListAsync();
            return View(users);
        }

        public async Task<IActionResult> AddUserToAdminRole(string email)
        {
            if (string.IsNullOrEmpty(email))
            {
                TempData["ErrorMessage"] = "Email is required";
                return RedirectToAction(nameof(UserManagement));
            }

            var user = await _userManager.FindByEmailAsync(email);
            if (user == null)
            {
                TempData["ErrorMessage"] = $"User with email {email} not found";
                return RedirectToAction(nameof(UserManagement));
            }

            var roleExists = await _roleManager.RoleExistsAsync("Admin");
            if (!roleExists)
            {
                await _roleManager.CreateAsync(new IdentityRole("Admin"));
            }

            if (await _userManager.IsInRoleAsync(user, "Admin"))
            {
                TempData["InfoMessage"] = $"User {email} is already in Admin role";
                return RedirectToAction(nameof(UserManagement));
            }

            var result = await _userManager.AddToRoleAsync(user, "Admin");
            if (!result.Succeeded)
            {
                string errors = string.Join(", ", result.Errors.Select(e => e.Description));
                TempData["ErrorMessage"] = $"Failed to add user to Admin role: {errors}";
                return RedirectToAction(nameof(UserManagement));
            }

            TempData["SuccessMessage"] = $"User {email} has been added to Admin role successfully";
            return RedirectToAction(nameof(UserManagement));
        }

        public async Task<IActionResult> AddUserToAdminRoleById(string userId)
        {
            if (string.IsNullOrEmpty(userId))
            {
                TempData["ErrorMessage"] = "User ID is required";
                return RedirectToAction(nameof(UserManagement));
            }

            var user = await _userManager.FindByIdAsync(userId);
            if (user == null)
            {
                TempData["ErrorMessage"] = $"User with ID {userId} not found";
                return RedirectToAction(nameof(UserManagement));
            }

            var roleExists = await _roleManager.RoleExistsAsync("Admin");
            if (!roleExists)
            {
                await _roleManager.CreateAsync(new IdentityRole("Admin"));
            }

            if (await _userManager.IsInRoleAsync(user, "Admin"))
            {
                TempData["InfoMessage"] = $"User {user.Email} is already in Admin role";
                return RedirectToAction(nameof(UserManagement));
            }
```

```
            var result = await _userManager.AddToRoleAsync(user, "Admin");
            if (!result.Succeeded)
            {
                string errors = string.Join(", ", result.Errors.Select(e => e.Description));
                TempData["ErrorMessage"] = $"Failed to add user to Admin role: {errors}";
                return RedirectToAction(nameof(UserManagement));
            }

            TempData["SuccessMessage"] = $"User {user.Email} has been added to Admin role successfully";
            return RedirectToAction(nameof(UserManagement));
        }

        [Authorize(Roles = "Admin")]
        public async Task<IActionResult> AddToAdminRoleDirect(string email)
        {
            if (string.IsNullOrEmpty(email))
            {
                TempData["ErrorMessage"] = "Email is required";
                return RedirectToAction(nameof(UserManagement));
            }

            var user = await _userManager.FindByEmailAsync(email);
            if (user == null)
            {
                TempData["ErrorMessage"] = $"User with email {email} not found";
                return RedirectToAction(nameof(UserManagement));
            }

            var adminRole = await _roleManager.FindByNameAsync("Admin");
            if (adminRole == null)
            {
                adminRole = new IdentityRole("Admin");
                await _roleManager.CreateAsync(adminRole);
            }

            bool isInRole = await _userManager.IsInRoleAsync(user, "Admin");
            if (isInRole)
            {
                TempData["InfoMessage"] = $"User {email} is already in Admin role";
                return RedirectToAction(nameof(UserManagement));
            }

            var result = await _userManager.AddToRoleAsync(user, "Admin");
            if (result.Succeeded)
            {
                TempData["SuccessMessage"] = $"User {email} has been added to Admin role successfully";
            }
            else
            {
                string errors = string.Join(", ", result.Errors.Select(e => e.Description));
                TempData["ErrorMessage"] = $"Failed to add user to Admin role: {errors}";
            }

            return RedirectToAction(nameof(UserManagement));
        }
    }
}
```

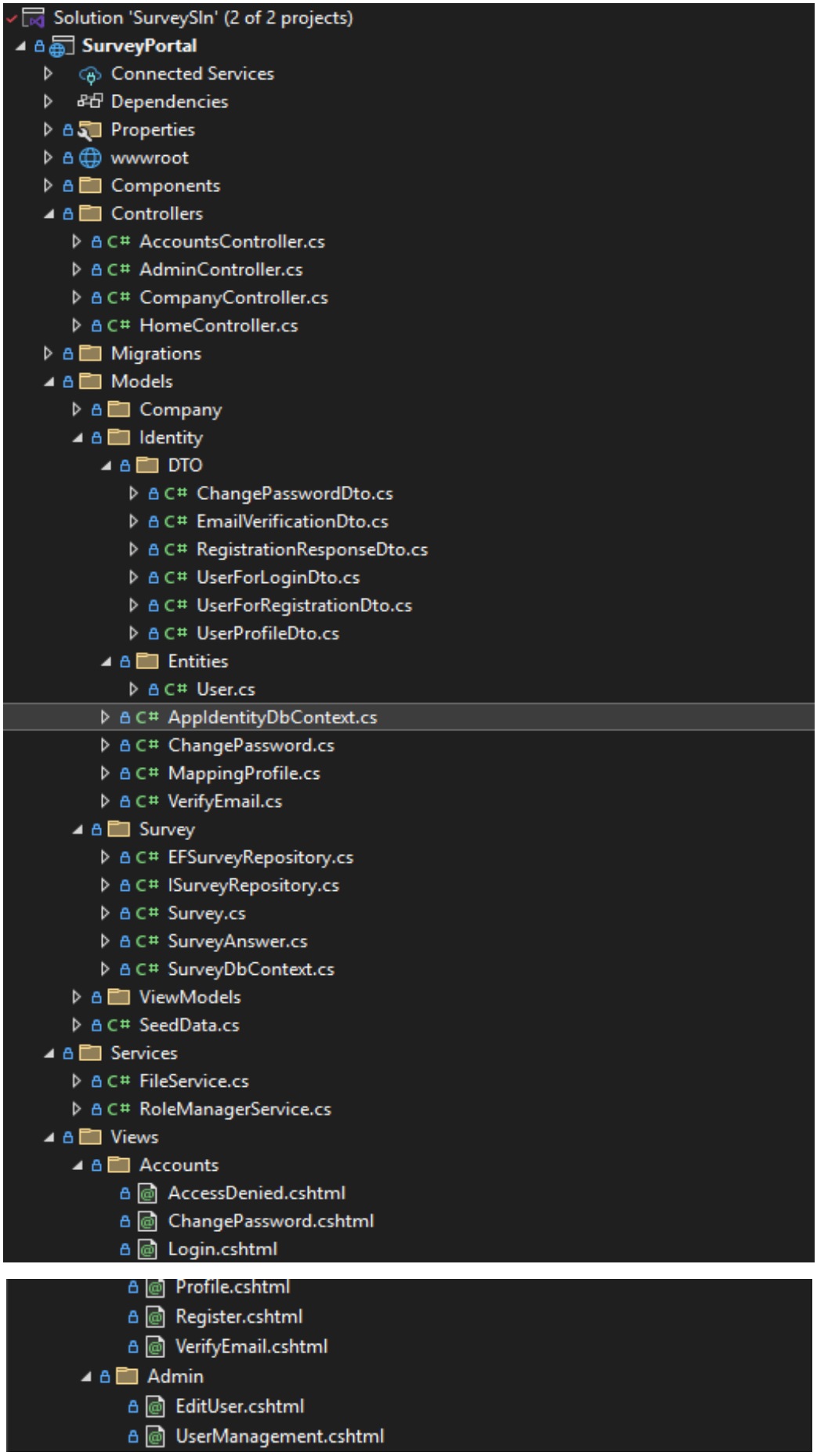

Рис. 1. Структура проекту після виконання лабораторної роботи.

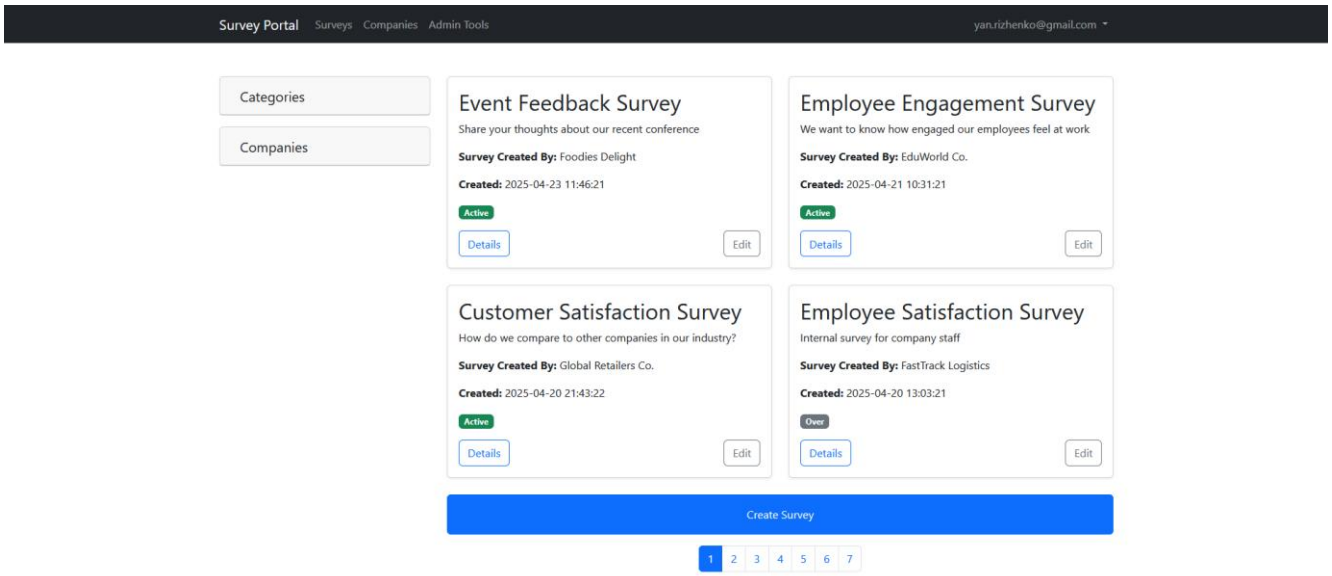

Рис. 2. Вигляд сайту після виконання лабораторної роботи (авторизований користувач має роль адмін).

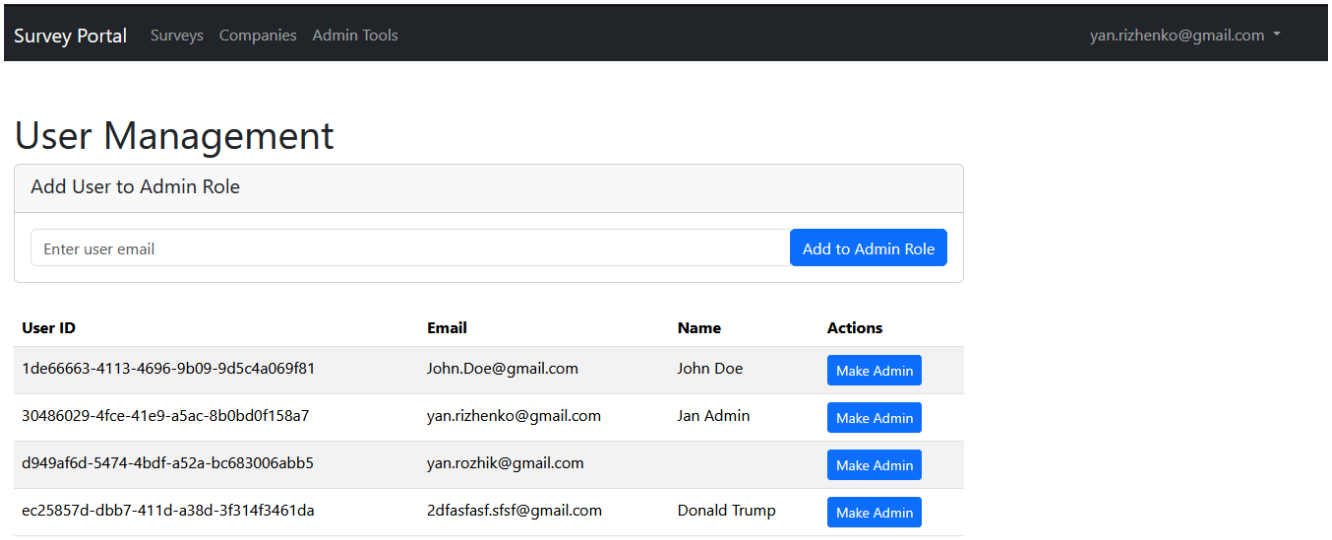

Рис. 3. View Admin/UserManagement.

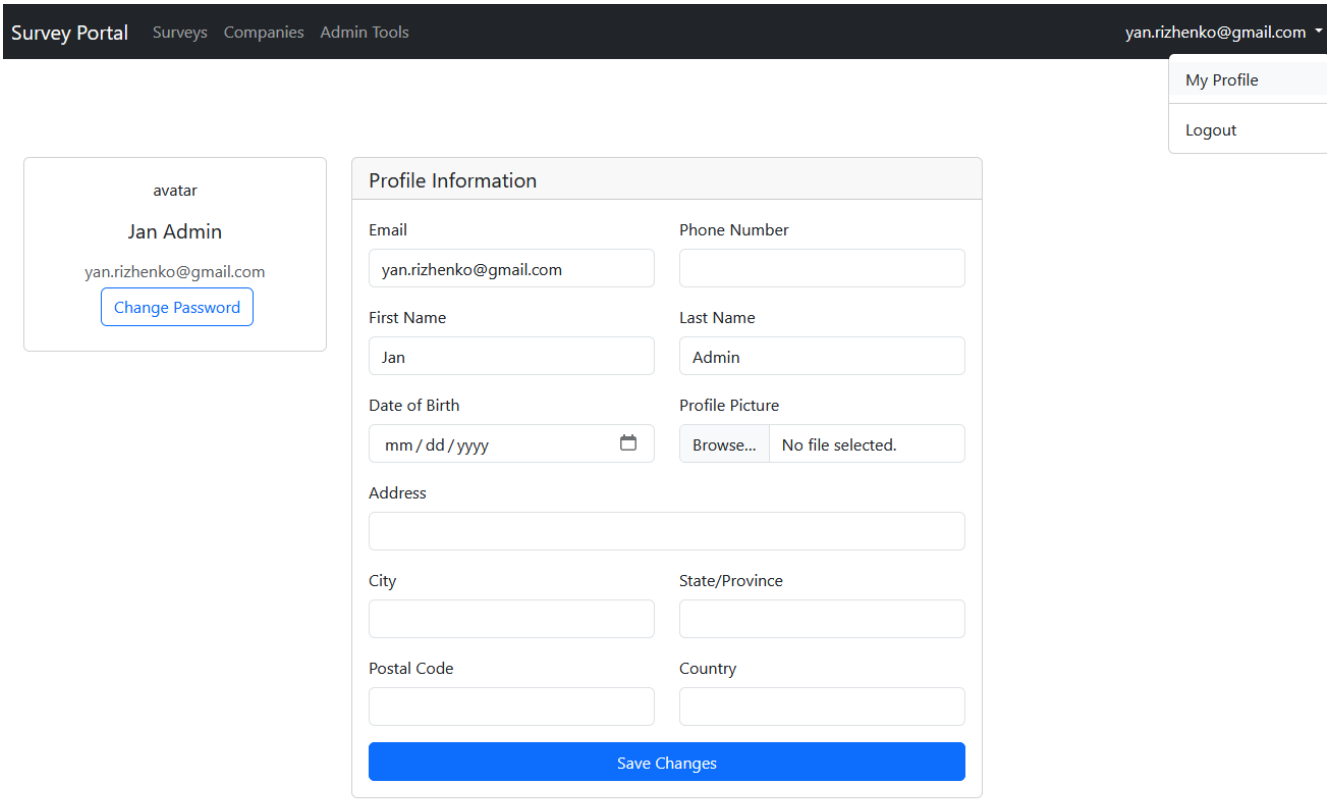

Рис. 3. View Accounts/Profile.

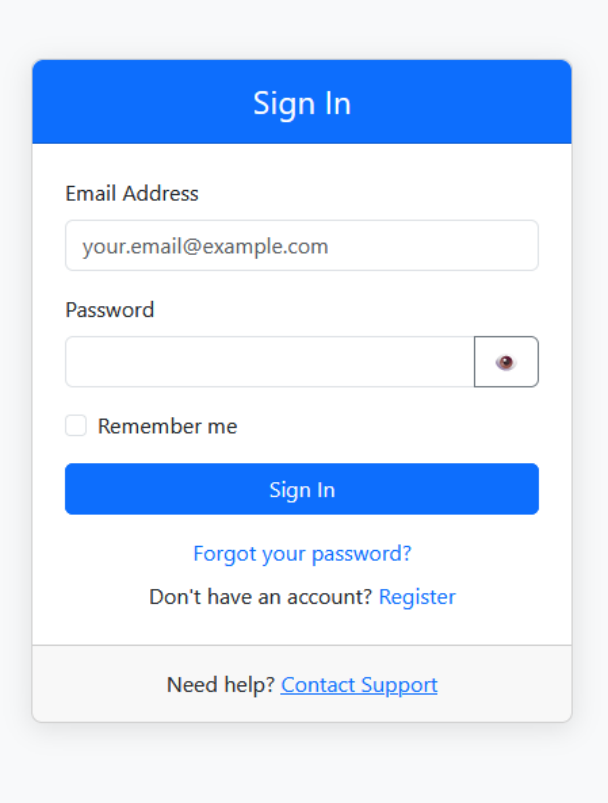

Рис. 4. View Accounts/Login.

Завдання 2-3.

В рішенні створити новий WebAPI проект.

Реалізуйте всі необхідні CRUD-операції в вашому WebAPI проекті, включаючи реєстрацію користувача. Перевірте працездатність ваших ендпоїнтів за допомогою Postman, Swagger або будь-яким іншим зручним інструментом.

Реалізуйте автентифікацію та авторизацію користувача з використанням механізму JSON Web Token (JWT). Вона повинна працювати наступним чином:

Клієнт відправляє запит на login-ендпоїнт зі своїм логіном та паролем.

Якщо користувач з таким логіном та паролем існує, тоді сервер повинен повернути згенерований JSON Web Token.

Для отримання доступу до дій, які вимагають авторизації, клієнт відправляє в запиті заголовок Authorization зі значенням Bearer TokenValue, де TokenValue – токен, який було згенеровано під час автентифікації.

Якщо токен валідний – сервер виконає запит, інакше поверне помилку 401 Unauthorized.

Листинг Program.cs після виконання лабораторної роботи:

```
using Microsoft.AspNetCore.Authentication.JwtBearer;
using Microsoft.AspNetCore.Identity;
using Microsoft.EntityFrameworkCore;
using Microsoft.IdentityModel.Tokens;
using System.Security.Cryptography;
using System.Text;
using WebAPI.Data;
using WebAPI.Models;

var builder = WebApplication.CreateBuilder(args);


builder.Services.AddControllers();
builder.Services.AddEndpointsApiExplorer();

builder.Services.AddDbContext<AppDbContext>(options =>
options.UseMySQL(builder.Configuration.GetConnectionString("Default")));

builder.Services.AddIdentity<IdentityUser, IdentityRole>(options =>
{
    options.Password.RequiredLength = 6;
    options.Password.RequireNonAlphanumeric = false;
    options.Password.RequireDigit = false;
    options.Password.RequireUppercase = false;
    options.Password.RequireLowercase = false;
})
    .AddEntityFrameworkStores<AppDbContext>()
    .AddDefaultTokenProviders();



builder.Services.AddAuthentication(options =>
{
    options.DefaultAuthenticateScheme = JwtBearerDefaults.AuthenticationScheme;
    options.DefaultChallengeScheme = JwtBearerDefaults.AuthenticationScheme;
    options.DefaultScheme = JwtBearerDefaults.AuthenticationScheme;
})
.AddJwtBearer(options =>
{
    options.TokenValidationParameters = new TokenValidationParameters
    {
        ValidateIssuer = true,
        ValidateAudience = false,
        ValidateLifetime = true,
```

```
        ValidateIssuerSigningKey = true,
        ValidIssuer = builder.Configuration["Jwt:Issuer"],
        IssuerSigningKey = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(builder.Configuration["Jwt:Key"]!))
    };
});

builder.Services.AddAuthorization(options =>
{
    options.AddPolicy("AdminPolicy", policy => policy.RequireRole("Admin"));
    options.AddPolicy("UserPolicy", policy => policy.RequireRole("User"));
});

builder.Logging.ClearProviders();
builder.Logging.AddConsole();


var app = builder.Build();

app.UseHttpsRedirection();

app.UseAuthentication();

app.UseAuthorization();

app.MapControllers();

app.Run();
```

Листинг appsettings.json після виконання лабораторної роботи:

```
{
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft.AspNetCore": "Warning"
    }
  },
  "Jwt": {
    "Issuer": "FreeTrained",
    "ExpiryMinutes": 3600,
    "Key": "Nf1fjv0TxOmTVrBbYFkFPwCeL4D9RGnPmgJj9/f7yqk="
  },
  "ConnectionStrings": {
    "Default": "server=localhost;database=webapidb;user=homeuser;password="
  },
  "AllowedHosts": "*"
}
```

Листинг AccountController після виконання лабораторної роботи:

```
using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Mvc;
using Microsoft.IdentityModel.Tokens;
using Microsoft.Win32;
using System.IdentityModel.Tokens.Jwt;
using System.Security.Claims;
using System.Text;
using WebAPI.Models;

namespace WebAPI.Controllers
{
    [Route("api/[controller]")]
    [ApiController]
    public class AccountController : ControllerBase
    {
        private readonly UserManager<IdentityUser> _userManager;
        private readonly RoleManager<IdentityRole> _roleManager;
        private readonly IConfiguration _configuration;

        public AccountController(UserManager<IdentityUser> userManager, RoleManager<IdentityRole> roleManager, IConfiguration
configuration)
        {
            _userManager = userManager;
            _roleManager = roleManager;
            _configuration = configuration;
        }

        [HttpPost("register")]
        public async Task<IActionResult> Register([FromBody] Register model)
        {
            var user = new IdentityUser { UserName = model.Username, Email = model.Email };
            var result = await _userManager.CreateAsync(user, model.Password);

            if (result.Succeeded)
            {
                await _userManager.AddToRoleAsync(user, "User");
                return Ok(new { message = "User registered successfully" });
            }

            return BadRequest(result.Errors);
        }
```

```
        [HttpPost("login")]
        public async Task<IActionResult> Login([FromBody] Login model)
        {
            var user = await _userManager.FindByNameAsync(model.Username);
            if (user != null && await _userManager.CheckPasswordAsync(user, model.Password))
            {
                var userRoles = await _userManager.GetRolesAsync(user);

                var authClaims = new List<Claim>
                {
                    new Claim(JwtRegisteredClaimNames.Sub, user.UserName!),
                    new Claim(JwtRegisteredClaimNames.Jti, Guid.NewGuid().ToString())
                };

                authClaims.AddRange(userRoles.Select(role => new Claim(ClaimTypes.Role, role)));

                var token = new JwtSecurityToken(
                    issuer: _configuration["Jwt:Issuer"],
                    expires: DateTime.Now.AddMinutes(double.Parse(_configuration["Jwt:ExpiryMinutes"]!)),
                    claims: authClaims,
                    signingCredentials: new SigningCredentials(new
SymmetricSecurityKey(Encoding.UTF8.GetBytes(_configuration["Jwt:Key"]!)),
                    SecurityAlgorithms.HmacSha256));

                return Ok(new { token = new JwtSecurityTokenHandler().WriteToken(token) });
            }

            return Unauthorized();
        }

        [HttpPost("add-role")]
        public async Task<IActionResult> AddRole([FromBody] string role)
        {
            if (!await _roleManager.RoleExistsAsync(role))
            {
                var result = await _roleManager.CreateAsync(new IdentityRole(role));
                if (result.Succeeded)
                {
                    return Ok(new { message = "Role added successfully" });
                }

                return BadRequest(result.Errors);
            }

            return BadRequest("Role already exists");
        }

        [HttpPost("assign-role")]
        public async Task<IActionResult> AssignRole([FromBody] UserRole model)
        {
            var user = await _userManager.FindByNameAsync(model.Username);
            if (user == null)
            {
                return BadRequest("User not found");
            }

            var result = await _userManager.AddToRoleAsync(user, model.Role);
            if (result.Succeeded)
            {
                return Ok(new { message = "Role assigned successfully" });
            }

            return BadRequest(result.Errors);
        }
    }
}
```

Листинг AdminController після виконання лабораторної роботи:

```
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;

namespace WebAPI.Controllers
{
    [Authorize(Roles = "Admin")]
    [Route("api/[controller]")]
    [ApiController]
    public class AdminController : Controller
    {
        [HttpGet]
        public IActionResult Get()
        {
            return Ok("You have accessed the Admin controller");
        }
    }
}
```

Листинг UserController після виконання лабораторної роботи:

```
using Microsoft.AspNetCore.Mvc;

namespace WebAPI.Controllers
{
    [Route("api/[controller]")]
    [ApiController]
    public class UserController : Controller
    {
        [HttpGet]
        public IActionResult Get()
        {
            return Ok("You have accessed the User controller");
        }
    }
}
```

Перевірка за допомогою Postman:

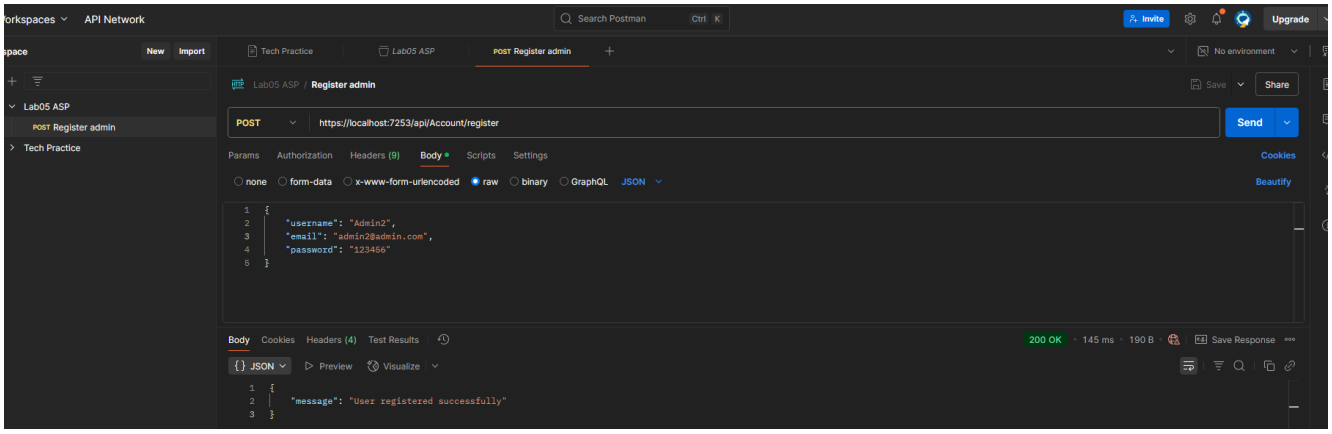

Рис. 5. Створення користувача Admin2.

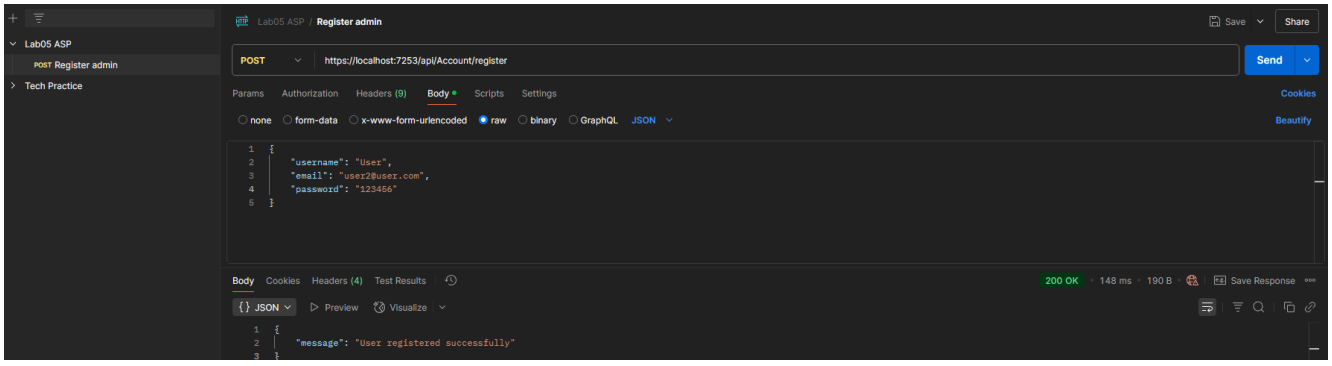

Рис. 6. Створення користувача User.

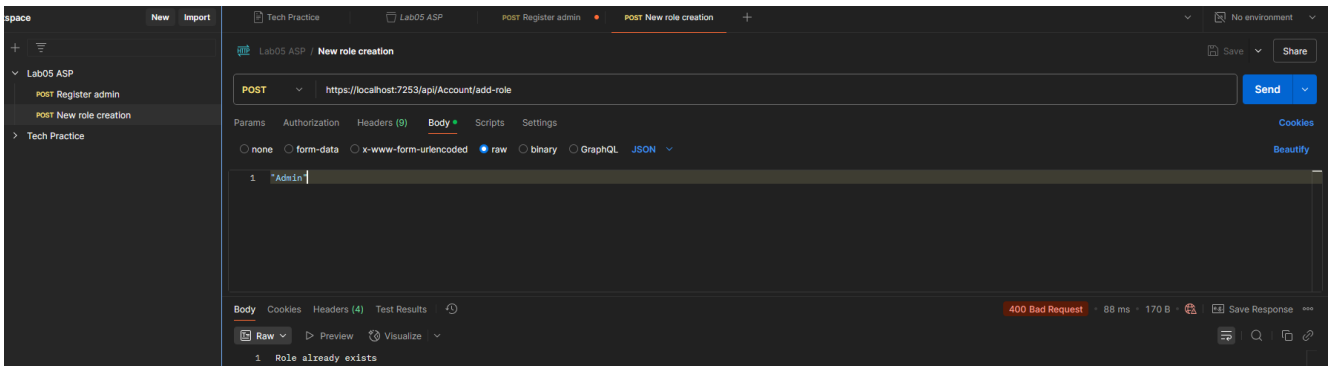

Рис. 7. Створення ролі Admin(вже створена).

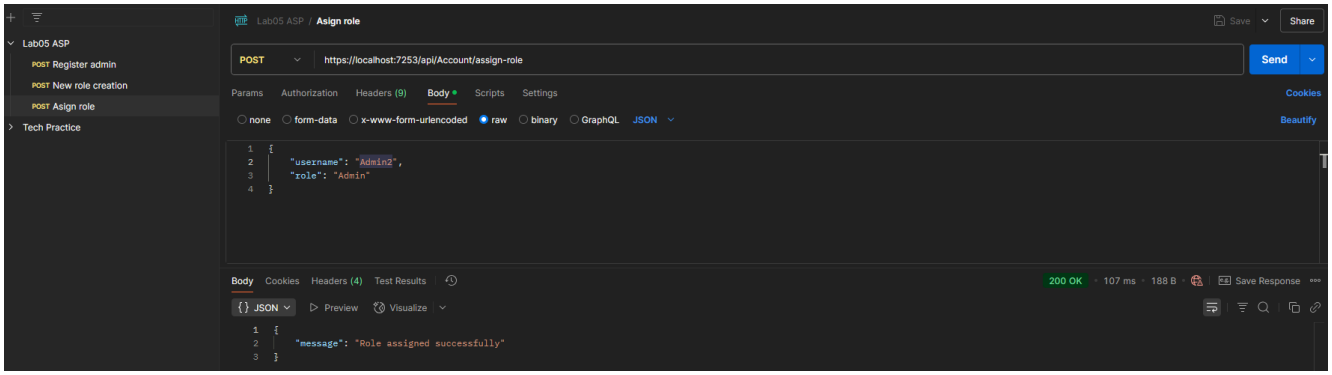

Рис. 8. Надання користувачу Admin2 ролі Admin.

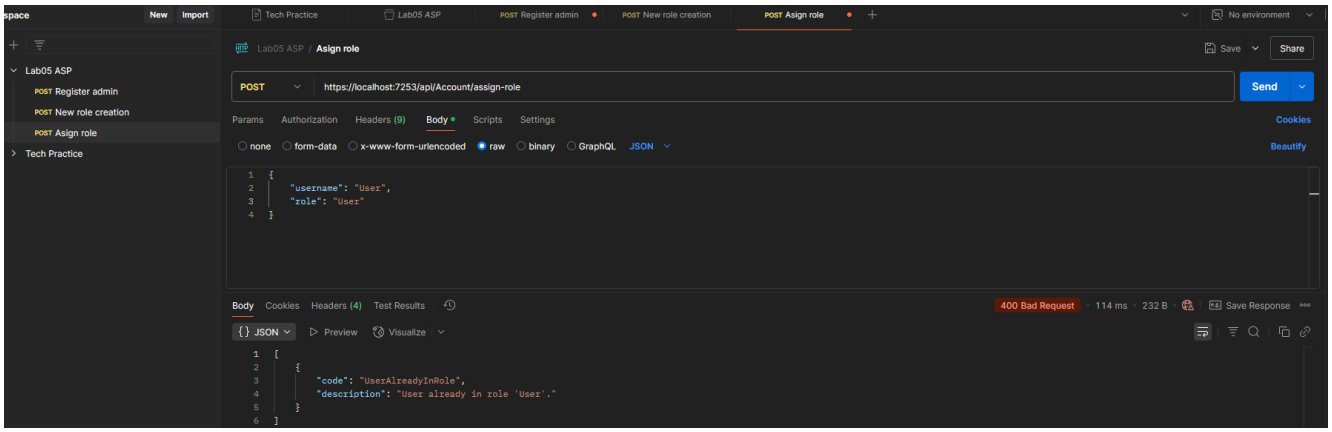

Рис. 9. Надання користувачу User ролі User(помилка – користувач вже має цю роль).

```
select * From aspnetusers
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

Id	UserName	NormalizedUserName	Email	NormalizedEmail	EmailConfirmed	PasswordHash	SecurityStamp	ConcurrencyStamp
1434b9f6-a2b5-4546-bfa4-db71c49b78c9	User	USER	user2@user.com	USER2@USER.COM	0	AQAAAAIAAYagAAAAEOaCTZy6z6MclymbhomIl...	BSGT3U2YXENTZB4KJYMBDVSMZSEYIPJ7	19f4038d-896d-4c0c-931c-5f66ece9f2bb
8241d2ef-eafc-4a88-9d48-4ecdc3f59917	Admin2	ADMIN2	admin2@admin.com	ADMIN2@ADMIN.COM	0	AQAAAAIAAYagAAAAENTaITLd1pmoiD9FGRnFn...	ZAS6LJT7SIJQSN7DOENNUHLJNL2KZNB7	b4a539bf-233c-4261-aa72-bb681e79af4c
addf92a7-a19c-4ad3-af0d-6821d5c3a813	Admin	ADMIN	admin@admin.com	ADMIN@ADMIN.COM	0	AQAAAAIAAYagAAAAEINxoyByix6Z+XgAIxYNY...	OKQ7ABVOKJAC3XJGOT5TAEAGZSOY5IJF	0f00e2ef-7767-4de8-8efb-b4bbf53e91e9

Рис. 10. Перевірка чи створено користувачів.

```
1    select * From aspnetroles
```

Result Grid | Filter Rows: | Edit: | Export/Import:

Id	Name	NormalizedName	ConcurrencyStamp
d3993498-bb8c-4c73-b502-26ed4ee92b92	User	USER	NULL
fa3387f8-9512-49b0-8a45-c71cf9e6343e	Admin	ADMIN	NULL

Рис. 10. Перевірка чи надано користувачам відповідні ролі.

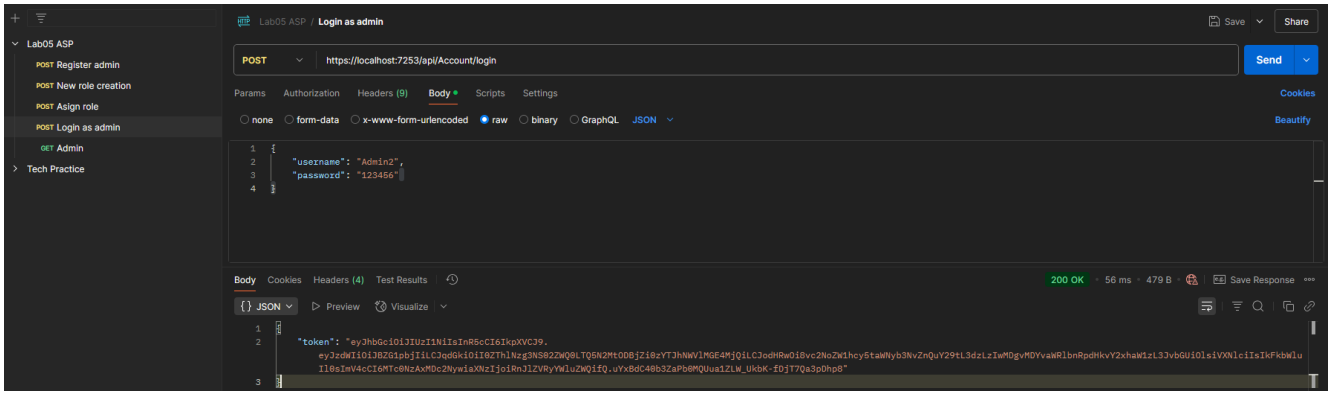

Рис. 11. Авторизація як адмін, отримуємо токен.

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiJBZG1pbjIiLCJqdGkiOiI0ZThlNzg3NS02ZWQ0LTQ5N2
MtODBjZi0zYTJhNWVlMGE4MjQiLCJodHRwOi8vc2NoZW1hcy5taWNyb3NvZnQuY29tL3dzLzIwMDgvMDYvaWRlbnRpd
HkvY2xhaW1zL3JvbGUiOlsiVXNlciIsIkFkbWluIl0sImV4cCI6MTc0NzAxMDc2NywiaXNzIjoiRnJlZVRyYWluZWQi
fQ.uYxBdC40b3ZaPb0MQUua1ZLW_UkbK-fDjT7Qa3pDhp8
```

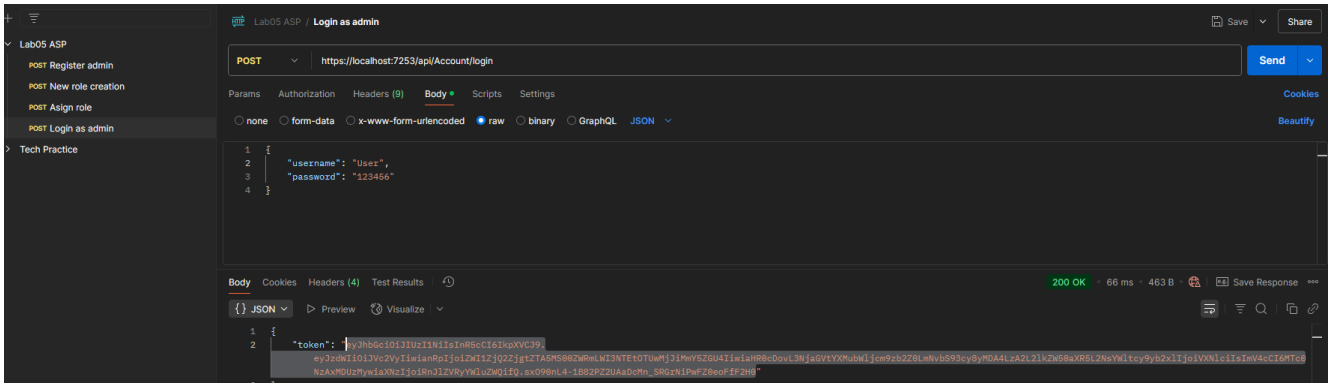

Рис. 12. Авторизація як юзер, отримуємо токен.


```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiJVc2VyIiwianRpIjoiZWI1ZjQ2ZjgtZTA5MS00ZWRmLW
I3NTEtOTUwMjJiMmY5ZGU4IiwiaHR0cDovL3NjaGVtYXMubWljcm9zb2Z0LmNvbS93cy8yMDA4LzA2L2lkZW50aXR5L
2NsYWltcy9yb2xlIjoiVXNlciIsImV4cCI6MTc0NzAxMDUzMywiaXNzIjoiRnJlZVRyYWluZWQifQ.sxO90nL4-
1B82PZ2UAaDcMn_SRGrNiPwFZ0eoFfF2H0
```


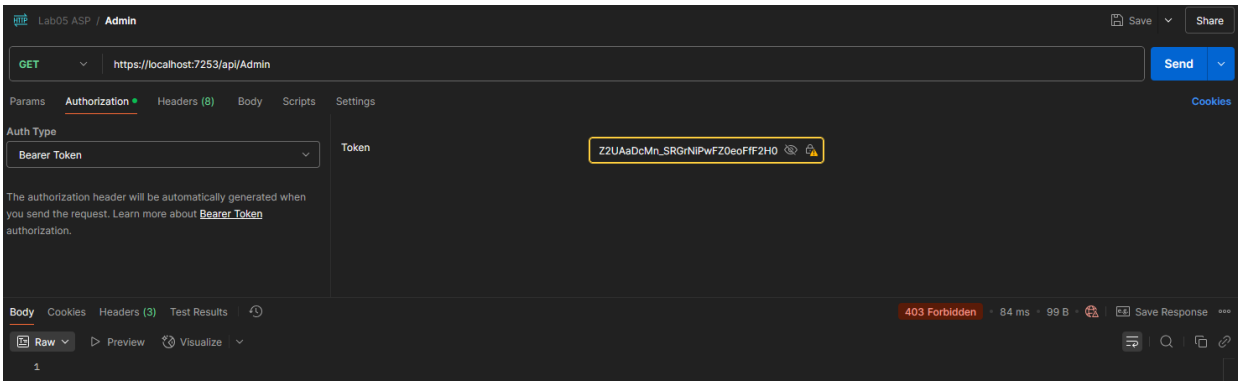

Рис. 13. Спроба увійти на сторінку адміністратора з токеном юзера – невдача.

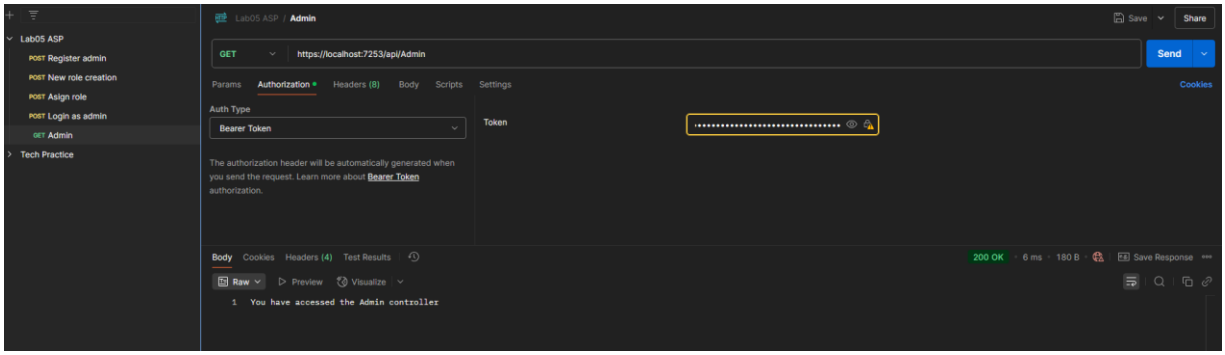

Рис. 14. Спроба увійти на сторінку адміністратора з токеном адміністратора – успіх.

Посилання на репозиторій: https://github.com/JanRizhenko/ASP.NET

Висновок: Аутентифікація та авторизація в ASP.NET дозволяють забезпечити захист ресурсів шляхом перевірки користувачів і контролю доступу до Web API. Використання JSON Web Token забезпечує безпечний та зручний спосіб передачі інформації про користувача між клієнтом і сервером.