ДОДАТОК А

Лістинг коду програми

```
Основа для ASP.NET Core веб-застосунку
using IndividualPersonalization.BLL.Infrastructure.Extensions;
using IndividualPersonalization.Infrastructure.Extensions;
using IndividualPersonalization.Infrastructure.GlobalFilters;
using Microsoft.AspNetCore.Authentication;
using Microsoft.EntityFrameworkCore;
using Microsoft.Extensions.Logging;
using System.IdentityModel.Tokens.Jwt;
using System.Text.Json.Serialization;

const string _defaultLoggerName = "logger";
NLog.ILogger logger = LogManager.Setup().LoadConfigurationFromFile("nlog.config").GetLogger(_defaultLoggerName);

try
{
    var builder = WebApplication.CreateBuilder(args);

    var connectionString = builder.Configuration.GetConnectionString("DefaultConnection");
    var mqttConfigurationSection = builder.Configuration.GetSection("MqttOptions");
    var projectConfigurationSection = builder.Configuration.GetSection("Project");

    builder.Services.ConfigureBLL(connectionString, mqttConfigurationSection, projectConfigurationSection);
    builder.Services.AddHostedServices();

    ConfigureNLog(builder);

    builder.Services.AddAuthentication()
        .AddIdentityServerJwt();

    JwtSecurityTokenHandler.DefaultInboundClaimTypeMap.Remove("role");

    builder.Services.AddLocalizations();
```

```
builder.Services.AddCors(options =>
{
    options.AddDefaultPolicy(
        builder =>
        {
            builder.AllowAnyOrigin()
                .AllowAnyHeader()
                .AllowAnyMethod();
        });
});

builder.Services.AddControllersWithViews(options =>
{
    options.Filters.Add<CustomAsyncAuthorizationFilter>();
    options.Filters.Add<CustomActionFilter>();
})
.AddJsonOptions(x =>
{
    x.JsonSerializerOptions.ReferenceHandler = ReferenceHandler.IgnoreCycles;
});

builder.Services.AddRazorPages();

#if DEBUG
    //builder.Services.AddSwaggerGen();
#endif

var app = builder.Build();

if (app.Environment.IsDevelopment())
{
    app.UseMigrationsEndPoint();
}
else
```

```
    {
        app.UseHsts();
    }

    app.UseRequestLocalization();

#if DEBUG
    //app.UseSwagger();
    //app.UseSwaggerUI();
#endif

    app.UseHttpsRedirection();
    app.UseStaticFiles();
    app.UseRouting();

    app.UseAuthentication();
    app.UseIdentityServer();
    app.UseAuthorization();
    app.UseCors();

    app.MapControllerRoute(
        name: "default",
        pattern: "{controller}/{action=Index}/{id?}");
    app.MapRazorPages();

    app.MapFallbackToFile("index.html"); ;

    app.Run();
}
catch (Exception exception)
{
    logger.Fatal(exception, "Stopped program because of exception");
    throw;
}
finally
```

```
{
    LogManager.Shutdown();
}

static void ConfigureNLog(WebApplicationBuilder builder)
{
    builder.Logging.ClearProviders();
    builder.Logging.SetMinimumLevel(Microsoft.Extensions.Logging.LogLevel.Trace);
    NLog.ILogger singletonlogger = LogManager.GetLogger(_defaultLoggerName);
    builder.Services.AddSingleton(singletonlogger);
}
```

Перевірка введених даних на валідність

```
using Microsoft.AspNetCore.Identity;

namespace IndividualPersonalization.DAL.Infrastructure.Localizers
{
    class LocalizedIdentityErrorDescriber : IdentityErrorDescriber
    {
        public override IdentityError DuplicateEmail(string email)
        {
            return new IdentityError
            {
                Code = nameof(DuplicateEmail),
                Description = string.Format("Електронна пошта '{0}' вже зайнята.", email)
            };
        }

        public override IdentityError DuplicateUserName(string userName)
        {
            return new IdentityError
            {
                Code = nameof(DuplicateUserName),
                Description = string.Format("Ім'я користувача '{0}' вже зайнято.", userName)
            };
        }
```

```
public override IdentityError InvalidEmail(string email)
{
    return new IdentityError
    {
        Code = nameof(InvalidEmail),
        Description = string.Format("Електронна адреса '{0}' недійсна.", email)
    };
}

public override IdentityError DuplicateRoleName(string role)
{
    return new IdentityError
    {
        Code = nameof(DuplicateRoleName),
        Description = string.Format("Ім'я ролі '{0}' вже зайнято.", role)
    };
}

public override IdentityError InvalidRoleName(string role)
{
    return new IdentityError
    {
        Code = nameof(InvalidRoleName),
        Description = string.Format("Недійсна назва ролі '{0}'.", role)
    };
}

public override IdentityError InvalidToken()
{
    return new IdentityError
    {
        Code = nameof(InvalidToken),
        Description = "Недійсний токен."
    };
```

```
        }

        public override IdentityError InvalidUserName(string userName)
        {
            return new IdentityError
            {
                Code = nameof(InvalidUserName),
                Description = string.Format("Ім'я користувача '{0}' недійсне, може містити лише літери або цифри.", userName)
            };
        }

        public override IdentityError LoginAlreadyAssociated()
        {
            return new IdentityError
            {
                Code = nameof(LoginAlreadyAssociated),
                Description = "Користувач з цим логіном вже існує."
            };
        }

        public override IdentityError PasswordMismatch()
        {
            return new IdentityError
            {
                Code = nameof(PasswordMismatch),
                Description = "Неправильний пароль."
            };
        }

        public override IdentityError PasswordRequiresDigit()
        {
            return new IdentityError
            {
                Code = nameof(PasswordRequiresDigit),
```

```
            Description = "Паролі повинні містити принаймні одну цифру ('0'-'9')."
        };
    }

    public override IdentityError PasswordRequiresLower()
    {
        return new IdentityError
        {
            Code = nameof(PasswordRequiresLower),
            Description = "Паролі повинні мати принаймні один нижній регістр ('a'-'z')."
        };
    }

    public override IdentityError PasswordRequiresNonAlphanumeric()
    {
        return new IdentityError
        {
            Code = nameof(PasswordRequiresNonAlphanumeric),
            Description = "Паролі повинні містити принаймні один небуквенно-цифровий
символ."
        };
    }

    public override IdentityError PasswordRequiresUniqueChars(int uniqueChars)
    {
        return new IdentityError
        {
            Code = nameof(PasswordRequiresUniqueChars),
            Description = string.Format("Пароль повинен мати {0} спеціальних символів.",
uniqueChars)
        };
    }

    public override IdentityError PasswordRequiresUpper()
    {
```

```
        return new IdentityError
        {
            Code = nameof(PasswordRequiresUpper),
            Description = "Паролі повинні мати принаймні один верхній регістр ('A'-'Z')."
        };
    }

    public override IdentityError PasswordTooShort(int length)
    {
        return new IdentityError
        {
            Code = nameof(PasswordTooShort),
            Description = string.Format("Паролі мають містити щонайменше {0} символів.", length)
        };
    }

    public override IdentityError UserAlreadyHasPassword()
    {
        return new IdentityError
        {
            Code = nameof(UserAlreadyHasPassword),
            Description = "Користувач уже має пароль."
        };
    }

    public override IdentityError UserAlreadyInRole(string role)
    {
        return new IdentityError
        {
            Code = nameof(UserAlreadyInRole),
            Description = string.Format("Користувач уже виконує роль '{0}'.", role)
        };
    }
```

```
public override IdentityError UserNotInRole(string role)
{
   return new IdentityError
   {
      Code = nameof(UserNotInRole),
      Description = string.Format("Користувач не виконує роль '{0}'.", role)
   };
}

public override IdentityError UserLockoutNotEnabled()
{
   return new IdentityError
   {
      Code = nameof(UserLockoutNotEnabled),
      Description = "Блокування не ввімкнено для цього користувача."
   };
}

public override IdentityError RecoveryCodeRedemptionFailed()
{
   return new IdentityError
   {
      Code = nameof(RecoveryCodeRedemptionFailed),
      Description = "Відновлення не вдалося."
   };
}

public override IdentityError ConcurrencyFailure()
{
   return new IdentityError
   {
      Code = nameof(ConcurrencyFailure),
      Description = "Оптимістична помилка паралельності, об'єкт було змінено."
   };
}
```

```
        public override IdentityError DefaultError()
        {
            return new IdentityError
            {
                Code = nameof(DefaultError),
                Description = "Сталася невідома помилка."
            };
        }
    }
}
```

Program.cs емулятора, яка відповідає за генерування даних з датчиків

```
using IndividualPersonalization.Emulator.Infrastructure;
using IndividualPersonalization.Emulator.Models;
using IndividualPersonalization.MQTT.Services.Other;
using Microsoft.Extensions.DependencyInjection;
using System.Text.Json;
using IndividualPersonalization.MQTT.Infrastructure;
using Microsoft.Extensions.Options;

var start = new DateTime(2024, 5, 15);
var step = TimeSpan.FromMinutes(1);

var devices = new List<Farm>()
{
    new(5),
    new(6)

};

var measurementConfigs = new List<MeasurementConfig>()
{
    new MeasurementConfig(MeasurementType.Temperature, 15, 35),
    new MeasurementConfig(MeasurementType.Humidity, 30, 50)
};
```

```
var serviceProvider = ServiceProviderBuilder.BuildServiceProvider();
var mqttService = serviceProvider.GetRequiredService<IMqttService>();
var options = serviceProvider.GetRequiredService<IOptions<ProjectOptions>>();

var mqttClient = await mqttService.CreateClientAsync();
var currentPosition = start;

while (true)
{
   foreach (var device in devices)
   {
      foreach (var measurementConfig in measurementConfigs)
      {
         var mesurement = measurementConfig.CreateValue(currentPosition);
         string message = JsonSerializer.Serialize(mesurement);
         await mqttService.PublishAsync(mqttClient,
device.CreateTopik(options.Value.Name, c => c.Id), message);
      }
   }

   if (currentPosition.Add(step) > DateTime.Now)
   {
      var wait = DateTime.Now - currentPosition.Add(step);
      await Task.Delay(wait);
   }
   else
   {
      await Task.Delay(TimeSpan.FromSeconds(30));
   }

   currentPosition = currentPosition.Add(step);
}
```

Налаштування моделі та взаємодії з базою даних

```
using IndividualPersonalization.DAL.EF.Common;
```

```
using IndividualPersonalization.DAL.Entities;
using IndividualPersonalization.DAL.Entities.Constants;
using IndividualPersonalization.DAL.Entities.Identity;
using IndividualPersonalization.DAL.Infrastructure.Extensions;
using Duende.IdentityServer.EntityFramework.Options;
using Microsoft.EntityFrameworkCore;
using Microsoft.Extensions.Options;
using System.Data;

namespace IndividualPersonalization.DAL.EF
{
    public class IndividualPersonalizationDbContext : CustomApiAuthorizationDbContext<User, Role, int, UserClaim, UserRole, UserLogin, RoleClaim, UserToken>
    {
        public DbSet<Farm> Farms { get; set; } = null!;
        public DbSet<Measurement> Measurements { get; set; } = null!;
        public DbSet<Template> Templates { get; set; } = null!;
        public DbSet<TemplateApplication> TemplateApplications { get; set; } = null!;

        public IndividualPersonalizationDbContext(DbContextOptions options, IOptions<OperationalStoreOptions> operationalStoreOptions)
            : base(options, operationalStoreOptions)
        {
            //if (Database.GetAppliedMigrations().Count() != Database.GetMigrations().Count())
            //{
            //    Database.Migrate();
            //}
        }

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

```
            builder.Entity<User>().HasMany(e => e.UserRoles).WithOne(e => e.User).HasForeignKey(ur => ur.UserId).IsRequired();
            builder.Entity<Role>().HasMany(e => e.UserRoles).WithOne(e => e.Role).HasForeignKey(ur => ur.RoleId).IsRequired();

            builder.Entity<Farm>().Navigation(a => a.User).AutoInclude();
            builder.Entity<Measurement>().Navigation(m => m.Farm).AutoInclude();
            builder.Entity<TemplateApplication>().Navigation(t => t.Template).AutoInclude();
            builder.Entity<TemplateApplication>().Navigation(t => t.Farm).AutoInclude();
            builder.Entity<Template>().Navigation(t => t.User).AutoInclude();
            builder.Entity<Template>().Navigation(t => t.TemplateApplications).AutoInclude();

            var roles = AddDefaultRoles(builder);
            AddDefaultAdmin(builder, roles.First(r => r.Name == RoleNames.Admin));
        }

        private void AddDefaultAdmin(ModelBuilder builder, Role adminRole)
        {
            //const string password = "!VwB/U73tcd_i!6";
            string passwordHash = "AQAAAAEAACcQAAAAEPDaIaHrB96A3kQU42tAfk94m+PEZeRmnI+dUTeRSshFeEp6qj7+94Q9pi0f21dT1w==";

            var admin = new User()
            {
                Id = 1,
                UserName = "admin@gmail.com",
                NormalizedUserName = "ADMIN@GMAIL.COM",
                Email = "admin@gmail.com",
                NormalizedEmail = "ADMIN@GMAIL.COM",
                PasswordHash = passwordHash,
                SecurityStamp = "OGV2JH6BM27RUQFSQDOXDM2WO7EN4F7S",
                ConcurrencyStamp = "562428c2-5d6b-40a9-a0a6-0665802ede1d"
            };
```

```
            var userRole = new UserRole()
            {
                RoleId = adminRole.Id,
                UserId = admin.Id
            };

            builder.Entity<User>().HasData(admin);
            builder.Entity<UserRole>().HasData(userRole);
        }

        private List<Role> AddDefaultRoles(ModelBuilder builder)
        {
            int counterId = 1;
            var roles = RoleNames.All.Select(roleName => new Role()
            {
                Id = counterId++,
                ConcurrencyStamp = roleName.NewGuid(),
                Name = roleName,
                NormalizedName = roleName.ToUpper()
            }).ToList();

            builder.Entity<Role>().HasData(roles);
            return roles;
        }
    }
}
```

Управління процесом аутентифікації в Angular додатку

```
import { Component, OnInit } from '@angular/core';
import { ActivatedRoute, Router } from '@angular/router';
import { BehaviorSubject } from 'rxjs';
import { ApplicationPaths, LoginActions, QueryParameterNames, ReturnUrlType } from 'src/infrastructure/api-authorization.constants';
import { AuthenticationResultStatus, AuthorizeService } from 'src/services/authorize.service';
```

```
@Component({
  selector: 'app-login',
  templateUrl: './login.component.html',
  styleUrls: ['./login.component.css']
})
export class LoginComponent implements OnInit {
  public message = new BehaviorSubject<string | null | undefined>(null);

  constructor(
    private authorizeService: AuthorizeService,
    private activatedRoute: ActivatedRoute,
    private router: Router) { }

  async ngOnInit() {
    const action = this.activatedRoute.snapshot.url[1];
    switch (action.path) {
      case LoginActions.Login:
        await this.login(this.getReturnUrl());
        break;
      case LoginActions.LoginCallback:
        await this.processLoginCallback();
        break;
      case LoginActions.LoginFailed:
        const message =
this.activatedRoute.snapshot.queryParamMap.get(QueryParameterNames.Message);
        this.message.next(message);
        break;
      case LoginActions.Profile:
        this.redirectToProfile();
        break;
      case LoginActions.Register:
        this.redirectToRegister();
        break;
      default:
```

```
      throw new Error(`Invalid action '${action}'`);
    }
  }


private async login(returnUrl: string): Promise<void> {
  const state: INavigationState = { returnUrl };
  const result = await this.authorizeService.signIn(state);
  this.message.next(undefined);
  switch (result.status) {
    case AuthenticationResultStatus.Redirect:
      break;
    case AuthenticationResultStatus.Success:
      await this.navigateToReturnUrl(returnUrl);
      break;
    case AuthenticationResultStatus.Fail:
      await this.router.navigate(ApplicationPaths.LoginFailedPathComponents, {
        queryParams: { [QueryParameterNames.Message]: result.message }
      });
      break;
    default:
      throw new Error(`Invalid status result ${(result as any).status}.`);
  }
}

private async processLoginCallback(): Promise<void> {
  const url = window.location.href;
  const result = await this.authorizeService.completeSignIn(url);
  switch (result.status) {
    case AuthenticationResultStatus.Redirect:
      // There should not be any redirects as completeSignIn never redirects.
      throw new Error('Should not redirect.');
    case AuthenticationResultStatus.Success:
      await this.navigateToReturnUrl(this.getReturnUrl(result.state));
      break;
```

```
            case AuthenticationResultStatus.Fail:
              this.message.next(result.message);
              break;
          }
        }

        private redirectToRegister(): any {
          this.redirectToApiAuthorizationPath(
            `${ApplicationPaths.IdentityRegisterPath}?returnUrl=${encodeURI('/' +
ApplicationPaths.Login)}`);
        }

        private redirectToProfile(): void {
          this.redirectToApiAuthorizationPath(ApplicationPaths.IdentityManagePath);
        }

        private async navigateToReturnUrl(returnUrl: string) {
          await this.router.navigateByUrl(returnUrl, {
            replaceUrl: true
          });
        }

        private getReturnUrl(state?: INavigationState): string {
          const fromQuery = (this.activatedRoute.snapshot.queryParams as
INavigationState).returnUrl;
          if (fromQuery &&
            !(fromQuery.startsWith(`${window.location.origin}/`) ||
              /\/[^\/].*/.test(fromQuery))) {
            throw new Error('Invalid return url. The return url needs to have the same origin as the
current page.');
          }
          return (state && state.returnUrl) ||
            fromQuery ||
            ApplicationPaths.DefaultLoginRedirectPath;
        }
```

```
  private redirectToApiAuthorizationPath(apiAuthorizationPath: string) {
    const redirectUrl = `${window.location.origin}/${apiAuthorizationPath}`;
    window.location.replace(redirectUrl);
  }
}

interface INavigationState {
  [ReturnUrlType]: string;
}
```

Сторінка перегляду ферм

```
<div class="container">
  <h1>{{'Farms' | translate}}</h1>

  <div class="mt-2 mb-2">
      <button class="btn btn-primary" [routerLink]="['/farms/new']">
        {{'AddNewFarm' | translate}}
      </button>
  </div>

  <table class="table">
    <thead>
        <tr>
            <th>{{'Name' | translate}}</th>
            <th>{{'Description' | translate}}</th>
            <th *ngIf="isAdmin | async">{{'User' | translate}}</th>
            <th>{{'Actions' | translate}}</th>
        </tr>
    </thead>
    <tbody>
        <tr *ngFor="let item of items | async">
            <td>{{item.name}}</td>
            <td>{{item.description}}</td>
            <td *ngIf="isAdmin | async">{{item.user?.email}}</td>
            <td class="row">
```

```
                <div  class="col-4">
                  <button class="btn btn-light" [routerLink]="['/farms/view/' + item.id]">{{'View' |
translate}}</button>
                </div>
                <div class="col-4">
                  <button class="btn btn-warning" [routerLink]="['/farms/' + item.id]">{{'Edit' |
translate}}</button>
                </div>
                <div class="col-4">
                  <button class="btn btn-danger" (click)="onDelete(item.id)">{{'Delete' |
translate}}</button>
                </div>
              </tr>
            </tbody>
          </table>
          </div>
```

Контролер для сутності ферм

```
          using Microsoft.AspNetCore.Authorization;
          using Microsoft.AspNetCore.Mvc;
          using IndividualPersonalization.BLL.Models.DTOs.Farms;
          using IndividualPersonalization.BLL.Models.Filters;
          using IndividualPersonalization.BLL.Services.Entities;
          using IndividualPersonalization.Infrastructure.Extensions;

          namespace IndividualPersonalization.Controllers
          {
            [ApiController]
            [Route("api/farms")]
            public class FarmsController : ControllerBase
            {
              protected readonly IFarmService _farmService;

              public FarmsController(IFarmService farmService)
              {
                _farmService = farmService;
```

```
        }

        [HttpGet]
        [Authorize]
        public async Task<IActionResult> FindByFilterAsync([FromQuery] FarmFilter filter)
        {
            return await _farmService.FindByFilterAsync(filter).ToResultAsync();
        }

        [HttpGet("{id}")]
        [Authorize]
        public async Task<IActionResult> FindByIdAsync([FromRoute] int id) //only own
        {
            return await _farmService.FindByIdAsync(id).ToResultAsync();
        }

        [HttpGet("{id}/info")]
        [Authorize]
        public async Task<IActionResult> FindInfoAsync([FromRoute] int id) //only own
        {
            return await _farmService.FindInfoAsync(id).ToResultAsync();
        }

        [HttpGet("{id}/chartDatas")]
        [Authorize]
        public async Task<IActionResult> FindChartDatasAsync([FromRoute] int id) //only
own
        {
            return await _farmService.FindChartDatasAsync(id).ToResultAsync();
        }

        [HttpPost]
        [Authorize]
        public async Task<IActionResult> CreateAsync([FromBody] FarmFormDTO hive)
        {
```

```
            return await _farmService.CreateAsync(hive).ToResultAsync();
        }

        [HttpPut]
        [Authorize]
        public async Task<IActionResult> UpdateAsync([FromBody] FarmFormDTO hive)
        {
            return await _farmService.UpdateAsync(hive).ToResultAsync();
        }

        [HttpDelete("{id}")]
        [Authorize]
        public async Task<IActionResult> DeleteAsync([FromRoute] int id) //only own
        {
            return await _farmService.DeleteAsync(id).ToResultAsync();
        }
    }
}
```

Контролер для сутності шаблонів

```
using IndividualPersonalization.BLL.Models.DTOs.Templates;
using IndividualPersonalization.BLL.Models.Filters;
using IndividualPersonalization.BLL.Services.Entities;
using IndividualPersonalization.Infrastructure.Extensions;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;

namespace IndividualPersonalization.Controllers
{
    [ApiController]
    [Route("api/templates")]
    public class TemplatesController : ControllerBase
    {
        protected readonly ITemplateService _templateService;

        public TemplatesController(ITemplateService templateService)
```

```
        {
            _templateService = templateService;
        }

        [HttpGet]
        [Authorize]
        public async Task<IActionResult> FindByFilterAsync([FromQuery] TemplateFilter
filter)
        {
            return await _templateService.FindByFilterAsync(filter).ToResultAsync();
        }

        [HttpGet("{id}")]
        [Authorize]
        public async Task<IActionResult> FindByIdAsync([FromRoute] int id) //only own
        {
            return await _templateService.FindByIdAsync(id).ToResultAsync();
        }

        [HttpPost]
        [Authorize]
        public async Task<IActionResult> CreateAsync([FromBody] TemplateFormDTO
model)
        {
            return await _templateService.CreateAsync(model).ToResultAsync();
        }

        [HttpPut]
        [Authorize]
        public async Task<IActionResult> UpdateAsync([FromBody] TemplateFormDTO
model)
        {
            return await _templateService.UpdateAsync(model).ToResultAsync();
        }
```

```
        [HttpDelete("{id}")]
        [Authorize]
        public async Task<IActionResult> DeleteAsync([FromRoute] int id) //only own
        {
            return await _templateService.DeleteAsync(id).ToResultAsync();
        }
    }
}
```

Контролер для сутності користувачів

```
using IndividualPersonalization.BLL.Models.Filters;
using IndividualPersonalization.BLL.Services.Entities;
using IndividualPersonalization.DAL.Entities.Constants;
using IndividualPersonalization.DAL.Entities.Identity;
using IndividualPersonalization.Infrastructure.Extensions;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;

namespace IndividualPersonalization.Controllers
{
    [ApiController]
    [Route("api/users")]
    public class UsersController : ControllerBase
    {
        protected readonly IUserService _userService;

        public UsersController(IUserService userService)
        {
            _userService = userService;
        }

        [HttpGet]
        [Authorize(Roles = RoleNames.Admin)]
        public async Task<IActionResult> FindByFilterAsync([FromQuery] UserFilter filter)
        {
            return await _userService.FindByFilterAsync(filter).ToResultAsync();
```

```
}

[HttpGet("{id}")]
[Authorize(Roles = RoleNames.Admin)]
public async Task<IActionResult> FindByIdAsync([FromRoute] int id)
{
   return await _userService.FindByIdAsync(id).ToResultAsync();
}

[HttpGet("currentProfile")]
[Authorize]
public async Task<IActionResult> FindCurrentProfileAsync()
{
   return await _userService.FindUserProfileAsync(User.Id()).ToResultAsync();
}

[HttpPost]
[Authorize(Roles = RoleNames.Admin)]
public async Task<IActionResult> CreateAsync([FromBody] User user)
{
   return await _userService.CreateAsync(user).ToResultAsync();
}

[HttpPut]
[Authorize(Roles = RoleNames.Admin)]
public async Task<IActionResult> UpdateAsync([FromBody] User user)
{
   return await _userService.UpdateAsync(user).ToResultAsync();
}

[HttpPut]
[Authorize(Roles = RoleNames.User)]
public async Task<IActionResult> UpdateOwnAsync([FromBody] User user)
{
   return await _userService.UpdateAsync(user).ToResultAsync();
```

```
        }

        [HttpDelete("{id}")]
        [Authorize(Roles = RoleNames.Admin)]
        public async Task<IActionResult> DeleteAsync([FromRoute] int id)
        {
            return await _userService.DeleteAsync(id).ToResultAsync();
        }
    }
}
```

ДОДАТОК Б

Демонстраційні матеріали

Формат	Позиція	ПОЗНАЧЕННЯ	НАЙМЕНУВАННЯ	Кількість	Примітки
			Текстові документи		
А4	1	ГЮИК. 505900.001 ПЗ	Пояснювальна	84	с.
			записка		
			Інші документи		
А4	2		Лістинг коду програми	26	с.
А4	3		Демонстраційний матеріал	14	с.

Змін	Арк.	№ докум.	Підпис	Дата	ГЮИК. 505900.001 ВД		
Розробив		Анісімова Д. М.			Розробка програмної системи для контролю параметрів культивування рослин у закритому просторі Відомість кваліфікаційної роботи	Літера	Аркуш 1 / Аркуш 1
Перевірив		Жарікова І. В.					
						Кафедра КІТАР ХНУРЕ	
Норм. контроль		Демська Н. П.					
Затв.		Невлюдов І. Ш.					