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.

Для встановлення Identity, в папці з проектом слід виконати наступну команду:

```
>>
PS D:\4 курс\ASP Net\Project\JobPortal> dotnet add package Microsoft.AspNetCore.Identity.EntityFrameworkCore --version 8.0.0
info :   OK https://api.nuget.org/v3/vulnerabilities/index.json 141ms https://api.nuget.org/v3/index.json to C:\Users\Admin\.nuget\packageroject\JobPortal\JobPortal.csproj'.csproj'.
info : Generating MSBuild file D:\4 курс\ASP Net\Project\JobPortal\obj\JobPortal.csproj.nuget.g.props.' added to file 'D:\4 курс\ASP Net\Pr
info : Writing assets file to disk. Path: D:\4 курс\ASP Net\Project\JobPortal\obj\project.assets.jsonbility.update.jsonms
log  : Restored D:\4 курс\ASP Net\Project\JobPortal\JobPortal.csproj (in 2,96 sec).ity.base.json 137mst.extensions.caching.memory.8.0.0.nup
PS D:\4 курс\ASP Net\Project\JobPortal> nerabilities/2025.11.06.05.42.07/2025.11.17.11.26.18/vulnerability.update.json 139ms
info : Package 'Microsoft.AspNetCore.Identity.EntityFrameworkCore' is compatible with all the specified frameworks in project 'D:\4 курс\AS
```

Рис. 1.1. Встановлення Identity

Далі слід створити клас AppIdentityDbContext з наступним вмістом:

Лістинг:

```
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Identity.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore;

namespace JobPortal.Data
{
    public class AppIdentityDbContext : IdentityDbContext<IdentityUser>
    {
        public AppIdentityDbContext(DbContextOptions<AppIdentityDbContext> options)
            : base(options)
        {
        }
    }
}
```

Далі слід додати нову ConnectionString в appsettings.json для бази даних, де зберігатимуться дані про користувачів.

Лістинг:

```
{
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft.AspNetCore": "Warning"
    }
  },
  "AllowedHosts": "*",
  "ConnectionStrings": {
    "JobPortalConnection":
"Server=(localdb)\\MSSQLLocalDB;Database=JobPortal;MultipleActiveResultSets=true",
    "IdentityConnection":
"Server=(localdb)\\MSSQLLocalDB;Database=JobPortalIdentity;MultipleActiveResultSets=
true"
  }
}
```

Сконфігурувати Identity в файлі Program.cs, додавши наступні рядки:

Лістинг:

```
using JobPortal.Data;
using JobPortal.Models;
using Microsoft.AspNetCore.Identity;
using Microsoft.EntityFrameworkCore;
using System.Reflection.Metadata.Ecma335;

namespace JobPortal
{
    public class Program
    {
        public static void Main(string[] args)
        {
            var builder = WebApplication.CreateBuilder(args);

            // Identity DB
            builder.Services.AddDbContext<AppIdentityDbContext>(options =>

options.UseSqlServer(builder.Configuration.GetConnectionString("IdentityConnection")
));

            // Identity Settings
            builder.Services.AddIdentity<IdentityUser, IdentityRole>(options =>
            {
                options.Password.RequiredLength = 8;
                options.Password.RequireUppercase = true;
                options.Password.RequireDigit = true;
                options.Password.RequireLowercase = true;
                options.Password.RequireNonAlphanumeric = false;

                options.User.RequireUniqueEmail = true;
            })
            .AddEntityFrameworkStores<AppIdentityDbContext>()
            .AddDefaultTokenProviders();

            ...

            var app = builder.Build();

            ...

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

            app.UseAuthentication();
```

```
            app.UseAuthorization();

            app.MapDefaultControllerRoute();

            ...

            app.Run();
        }
    }
}
```

Створіть міграцію для бази даних Identity наступною командою. Та застосуйте її за допомогою наступної команди:

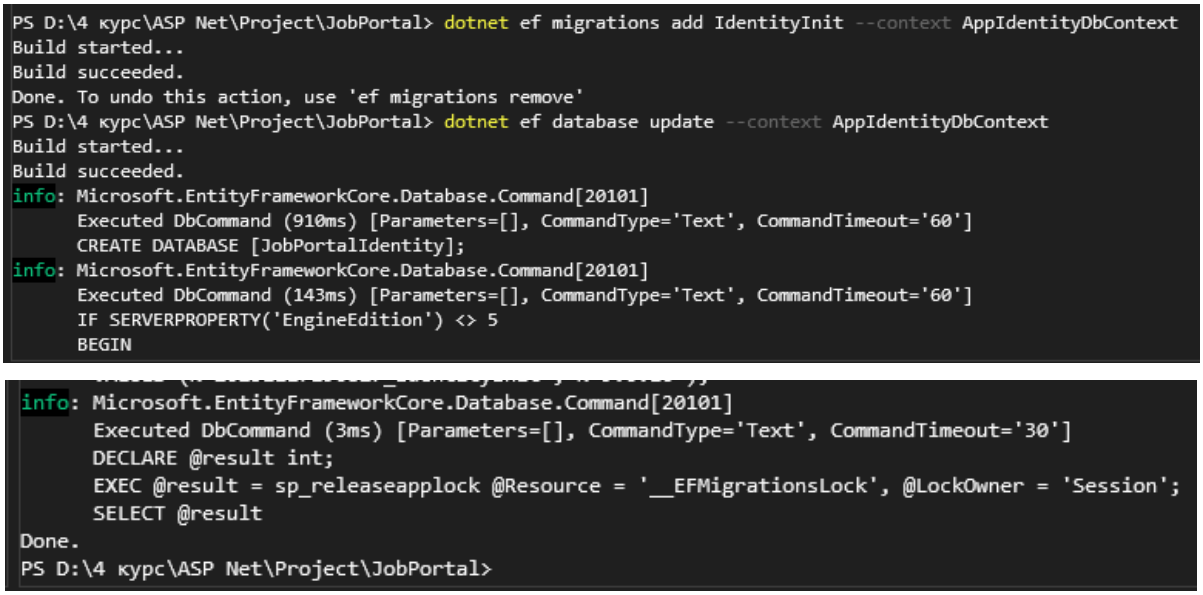

Рис. 1.2-3. Створення міграції та застосування змін у БД

Після конфігурації Identity, реалізуйте наступні функціональності:

1. Реєстрація користувача (приклад)

a. встановити вимогу по унікальності електронної адреси користувача

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

c. реалізувати поле для підтвердження введеного паролю

Лістинг контролера AccountController.cs:

```
using JobPortal.Models.ViewModels;
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Authorization;

namespace JobPortal.Controllers
{
    public class AccountController : Controller
    {
        private readonly UserManager<IdentityUser> userManager;
        private readonly SignInManager<IdentityUser> signInManager;
```

```
        public AccountController(UserManager<IdentityUser> um,
    SignInManager<IdentityUser> sm)
        {
            userManager = um;
            signInManager = sm;
        }

        public IActionResult Register() => View();

        [HttpPost]
        public async Task<IActionResult> Register(RegisterViewModel model)
        {
            if (ModelState.IsValid)
            {
                var user = new IdentityUser
                {
                    Email = model.Email,
                    UserName = model.UserName
                };

                var result = await userManager.CreateAsync(user, model.Password);

                if (result.Succeeded)
                {
                    await userManager.AddToRoleAsync(user, "User");
                    await signInManager.SignInAsync(user, false);
                    return RedirectToAction("Index", "Home");
                }

                foreach (var error in result.Errors)
                    ModelState.AddModelError("", error.Description);
            }
            return View(model);
        }

        public IActionResult Login(string? returnUrl) =>
            View(new LoginViewModel { ReturnUrl = returnUrl });

        [HttpPost]
        public async Task<IActionResult> Login(LoginViewModel model)
        {
            if (ModelState.IsValid)
            {
                var user = await userManager.FindByNameAsync(model.UserName);

                if (user != null)
                {
                    var result = await signInManager.PasswordSignInAsync(user,
                        model.Password, false, false);

                    if (result.Succeeded)
                        return Redirect(model.ReturnUrl ?? "/");
                }

                ModelState.AddModelError("", "Невірний логін або пароль");
            }
```

```
            return View(model);
        }

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

        [Authorize]
        public IActionResult Profile()
        {
            return View();
        }
    }
}
```

Лістинг моделі для форми входу LoginViewModel:

```
using System.ComponentModel.DataAnnotations;

namespace JobPortal.Models.ViewModels
{
    public class LoginViewModel
    {
        [Required]
        public string UserName { get; set; } = "";

        [Required, DataType(DataType.Password)]
        public string Password { get; set; } = "";

        public string? ReturnUrl { get; set; }
    }
}
```

Лістинг моделі для форми реєстрації RegisterViewModel:

```
using System.ComponentModel.DataAnnotations;

namespace JobPortal.Models.ViewModels
{
    public class RegisterViewModel
    {
        [Required, EmailAddress]
        public string Email { get; set; } = "";

        [Required]
        public string UserName { get; set; } = "";

        [Required, DataType(DataType.Password)]
        [MinLength(8)]
        public string Password { get; set; } = "";

        [Required, DataType(DataType.Password)]
        [Compare("Password", ErrorMessage = "Passwords do not match")]
        public string ConfirmPassword { get; set; } = "";
    }
}
```

2. Автентифікація користувача (Login)

Лістинг Views/Account/Login:

```
@model JobPortal.Models.ViewModels.LoginViewModel

<h2>Вхід</h2>

<form asp-action="Login" method="post">
    <div asp-validation-summary="All"></div>

    <label>Ім’я користувача</label>
    <input asp-for="UserName" class="form-control" />

    <label>Пароль</label>
    <input asp-for="Password" class="form-control" />

    <button class="btn btn-primary mt-3">Увійти</button>
</form>
```

3. Вихід користувача (Logout)

Лістинг:

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

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

Лістинг Views/Account/Profile:

```
@model JobPortal.Models.ViewModels.ProfileViewModel

<h2 class="mb-4">Особистий кабінет</h2>

<div class="card shadow-sm">
    <div class="card-body">

        <h4 class="mb-3">Ваші дані</h4>

        <form asp-action="UpdateProfile" method="post">
            <input type="hidden" asp-for="Id" />

            <div class="mb-3">
                <label class="form-label">Повне ім’я</label>
                <input asp-for="FullName" class="form-control" />
            </div>

            <div class="mb-3">
                <label class="form-label">Email</label>
                <input asp-for="Email" class="form-control" />
            </div>

            <div class="mb-3">
```

```
                <label class="form-label">Роль</label>
                <div class="form-control bg-light">@Model.Role</div>
            </div>

            <div class="mb-3">
                <label class="form-label">Дата створення</label>
                <div class="form-control bg-light">
                    @Model.CreatedAt.ToString("dd.MM.yyyy HH:mm")
                </div>
            </div>

            <button class="btn btn-primary" type="submit">Зберегти зміни</button>
            <a asp-action="Logout" class="btn btn-danger ms-2">Вийти</a>
        </form>

    </div>
</div>
```

Завдання на додаткові бали (5 балів): реалізувати роботу з ролями користувачів.

Лістинг Data/RoleSeeder.cs:

```
using Microsoft.AspNetCore.Identity;

namespace JobPortal.Data
{
 public static class RoleSeeder
 {
       public static async Task SeedAsync(RoleManager<IdentityRole> rm)
       {
             if (!await rm.RoleExistsAsync("User"))
                   await rm.CreateAsync(new IdentityRole("User"));

             if (!await rm.RoleExistsAsync("Employer"))
                   await rm.CreateAsync(new IdentityRole("Employer"));
       }
 }
}
```

Слід також змінити доступ для користувачів з різними ролями.

Лістинг ApplicationController.cs:

```
using JobPortal.Models;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using Newtonsoft.Json;

public class ApplicationController : Controller
{
    private readonly IPortalRepository _repository;

    public ApplicationController(IPortalRepository repo)
    {
        _repository = repo;
    }

    // ======= EMPLOYER AREA =======
```

```
        [Authorize(Roles = "Employer")]
        public async Task<IActionResult> Index()
        {
            ...
        }

        [Authorize(Roles = "Employer")]
        public async Task<IActionResult> Details(int? id)
        {
            if (id == null) return NotFound();

            var application = await _repository.Applications
                .Include(a => a.Job)
                .FirstOrDefaultAsync(m => m.Id == id);

            return application == null ? NotFound() : View(application);
        }

        // EMPLOYER ONLY CRUD
        [Authorize(Roles = "Employer")]
        public IActionResult Create()
        {
            ViewBag.Jobs = _repository.Jobs.ToList();
            return View("Edit", new Application());
        }

        [Authorize(Roles = "Employer")]
        [HttpPost, ValidateAntiForgeryToken]
        public IActionResult Create(Application app)
        {
            ...
        }

        [Authorize(Roles = "Employer")]
        public IActionResult Edit(int id)
        {
            ...
        }

        [Authorize(Roles = "Employer")]
        [HttpPost, ValidateAntiForgeryToken]
        public IActionResult Edit(Application app)
        {
            ...
        }

        [Authorize(Roles = "Employer")]
        public IActionResult Delete(int id)
        {
            ...
        }

        [Authorize(Roles = "Employer")]
        [HttpPost, ActionName("Delete"), ValidateAntiForgeryToken]
        public IActionResult DeleteConfirmed(int id)
        {
            ...
        }


        // ======= CANDIDATE ZONE =======

        [Authorize(Roles = "Candidate")]
        public IActionResult Apply()
        {
            ...
```

```
    }


    [Authorize(Roles = "Candidate")]
    [HttpPost]
    public IActionResult Apply(ApplicationForm model)
    {
        ...
    }

    [Authorize(Roles = "Candidate")]
    public IActionResult Submit()
    {
        ...
    }

    private const string SessionKey = "ApplicationForm";
}
```

Лістинг Data/RoleSeeder.cs:

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

namespace JobPortal.Controllers
{
    public class JobsController : Controller
    {
        private readonly IPortalRepository _repository;

        public JobsController(IPortalRepository repo)
        {
            _repository = repo;
        }

        // ---------- PUBLIC (доступні всім) ----------
        [AllowAnonymous]
        public IActionResult Index() => View(_repository.Jobs);

        [AllowAnonymous]
        public IActionResult Details(int id)
        {
            var job = _repository.GetJobById(id);
            if (job == null) return NotFound();
            return View(job);
        }

        // ---------- EMPLOYER ONLY ----------
        [Authorize(Roles = "Employer")]
        public IActionResult Create()
        {
            return View("Edit", new Job());
        }

        [Authorize(Roles = "Employer")]
        [HttpPost]
        [ValidateAntiForgeryToken]
        public IActionResult Create(Job job)
        {
            ...
        }

        [Authorize(Roles = "Employer")]
        public IActionResult Edit(int id)
```

```
        {
            var job = _repository.GetJobById(id);
            if (job == null) return NotFound();
            return View(job);
        }

        [Authorize(Roles = "Employer")]
        [HttpPost]
        [ValidateAntiForgeryToken]
        public IActionResult Edit(Job job)
        {
            ...
        }

        [Authorize(Roles = "Employer")]
        public IActionResult Delete(int id)
        {
            ...
        }

        [Authorize(Roles = "Employer")]
        [HttpPost, ActionName("Delete")]
        [ValidateAntiForgeryToken]
        public IActionResult DeleteConfirmed(int id)
        {
            ...
        }
    }
}
```

Результат:

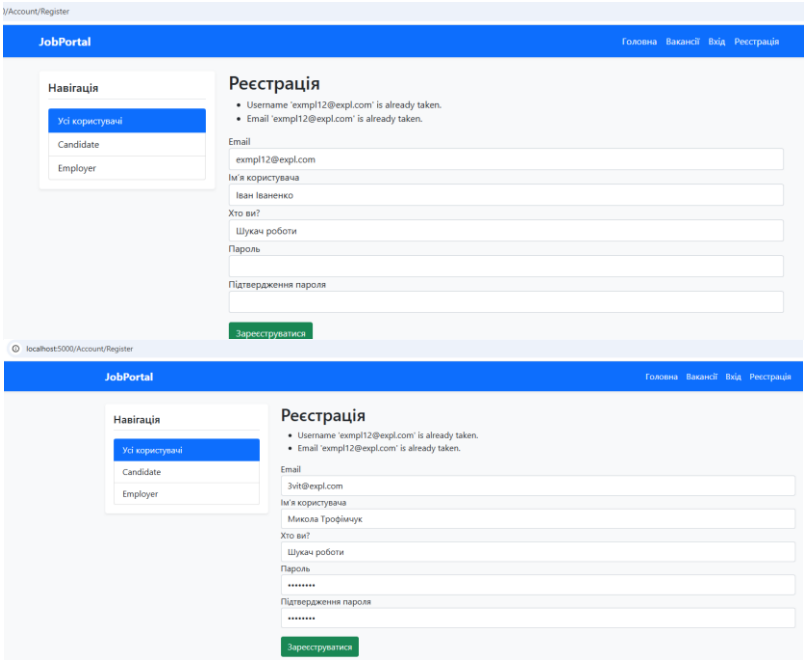

Рис. 1.4. Реєстрація користувача (шукач роботи)

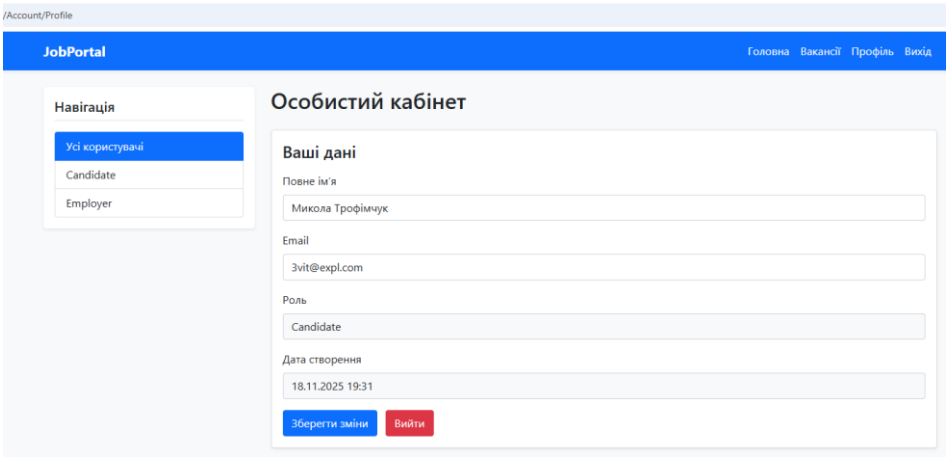

Рис. 1.5. Профіль користувача (з можливістю редагування та виходу з аккаунту)

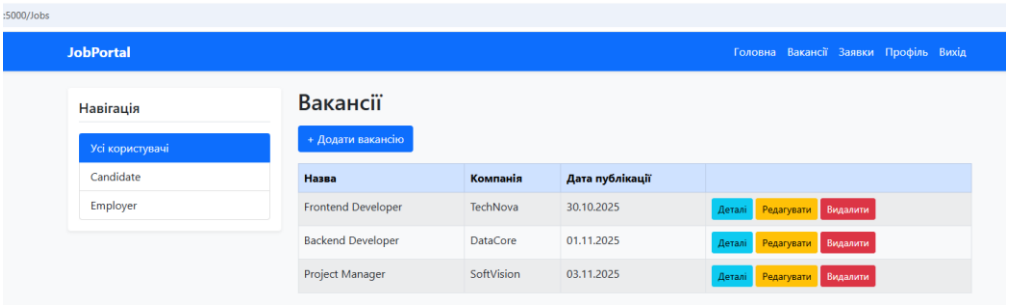

Рис. 1.6. Вхід до профілю користувача з роллю роботодавця (доступні заявки та редагування вакансій)

Завдання 2.

В рішенні створити новий WebAPI проект. Цей проект повинен реалізовувати всі функціональності, які наявні в MVC проекті у вигляді REST-ендпоїнтів. Для кращої організації і перевикористання коду слід перенести файли для роботи з БД (репозиторії, контексти, моделі) в окрему бібліотеку і додати посилання на неї в MVC та WebAPI проектах. Встановіть необхідні залежності для проекту бібліотеки.

Після цього налаштуйте ConnectionString'и для створених БД для WebAPI проекту в файлі appsettings.json. Таким чином, після перенесення файлів для роботи з БД в окрему бібліотеку ми можемо використовувати всі необхідні сервіси в обох проектах.

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.

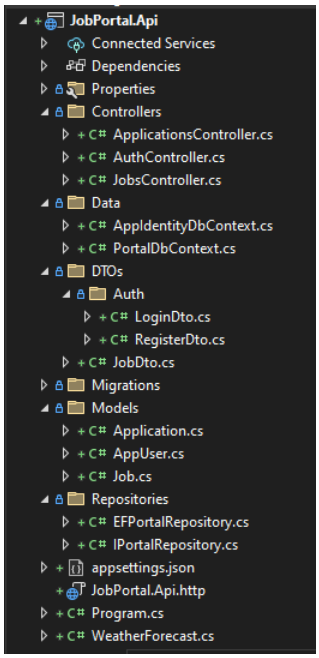

Рис. 2.1. Створений проект

Лістинг AuthController.cs:

```
using Microsoft.AspNetCore.Mvc;
using JobPortal.Api.DTOs.Auth;
using JobPortal.Api.Models;
using Microsoft.AspNetCore.Identity;
using Microsoft.IdentityModel.Tokens;
using System.IdentityModel.Tokens.Jwt;
using System.Text;
using System.Security.Claims;

namespace JobPortal.Api.Controllers
{
    [ApiController]
    [Route("api/account")]
    public class AuthController : ControllerBase
    {
        private readonly UserManager<AppUser> _userManager;
        private readonly SignInManager<AppUser> _signInManager;
        private readonly IConfiguration _config;
        private readonly RoleManager<IdentityRole> _roleManager;

        public AuthController(UserManager<AppUser> userManager,
SignInManager<AppUser> signInManager,
            IConfiguration config, RoleManager<IdentityRole> roleManager)
        {
            _userManager = userManager;
            _signInManager = signInManager;
            _config = config;
            _roleManager = roleManager;
        }

        [HttpPost("register")]
        public async Task<IActionResult> Register(RegisterDto dto)
        {
            if (!ModelState.IsValid) return BadRequest(ModelState);
```

```
            var user = new AppUser
            {
                UserName = dto.Email,
                Email = dto.Email,
                FullName = dto.FullName
            };

            var result = await _userManager.CreateAsync(user, dto.Password);
            if (!result.Succeeded) return BadRequest(result.Errors);

            // ensure role exists
            if (!await _roleManager.RoleExistsAsync(dto.Role))
                await _roleManager.CreateAsync(new IdentityRole(dto.Role));

            await _userManager.AddToRoleAsync(user, dto.Role);

            return Ok(new { message = "Registered" });
        }

        [HttpPost("login")]
        public async Task<IActionResult> Login(LoginDto dto)
        {
            var user = await _userManager.FindByEmailAsync(dto.Email);
            if (user == null) return Unauthorized();

            var    chk    =    await    _signInManager.CheckPasswordSignInAsync(user,
dto.Password, false);
            if (!chk.Succeeded) return Unauthorized();

            var roles = await _userManager.GetRolesAsync(user);

            var claims = new List<Claim>
            {
                new Claim(ClaimTypes.NameIdentifier, user.Id),
                new Claim(ClaimTypes.Name, user.UserName ?? user.Email ?? ""),
                new Claim(ClaimTypes.Email, user.Email ?? "")
            };
            foreach (var r in roles) claims.Add(new Claim(ClaimTypes.Role, r));

            var                     key                     =                     new
SymmetricSecurityKey(Encoding.UTF8.GetBytes(_config["Jwt:Key"]));
            var creds = new SigningCredentials(key, SecurityAlgorithms.HmacSha256);
            var token = new JwtSecurityToken(
                issuer: _config["Jwt:Issuer"],
                audience: _config["Jwt:Audience"],
                claims: claims,
                expires:
DateTime.UtcNow.AddMinutes(double.Parse(_config["Jwt:DurationMinutes"])),
                signingCredentials: creds
            );

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

Лістинг ApplicationController.cs:

```
using Microsoft.AspNetCore.Mvc;
using JobPortal.Api.Repositories;
using JobPortal.Api.Models;
using Microsoft.AspNetCore.Authorization;

namespace JobPortal.Api.Controllers
```

```
{
    [ApiController]
    [Route("api/[controller]")]
    public class ApplicationsController : ControllerBase
    {
        private readonly IPortalRepository _repo;
        public ApplicationsController(IPortalRepository repo) { _repo = repo; }

        [HttpGet]
        [Authorize(Roles = "Employer")]
        public IActionResult GetAll() => Ok(_repo.Applications.ToList());

        [HttpGet("{id}")]
        [Authorize(Roles = "Employer")]
        public IActionResult Get(int id)
        {
            var app = _repo.GetApplicationById(id);
            if (app == null) return NotFound();
            return Ok(app);
        }

        [HttpPost]
        [Authorize(Roles = "Candidate")]
        public IActionResult Create(Application app)
        {
            _repo.CreateApplication(app);
            return CreatedAtAction(nameof(Get), new { id = app.Id }, app);
        }

        [HttpDelete("{id}")]
        [Authorize(Roles = "Employer")]
        public IActionResult Delete(int id)
        {
            var app = _repo.GetApplicationById(id);
            if (app == null) return NotFound();
            _repo.DeleteApplication(app);
            return NoContent();
        }
    }
}
```

Лістинг JobController.cs:

```
using Microsoft.AspNetCore.Mvc;
using JobPortal.Api.Repositories;
using JobPortal.Api.Models;
using JobPortal.Api.DTOs;
using Microsoft.AspNetCore.Authorization;

namespace JobPortal.Api.Controllers
{
    [ApiController]
    [Route("api/[controller]")]
    public class JobsController : ControllerBase
    {
        private readonly IPortalRepository _repo;
        public JobsController(IPortalRepository repo) { _repo = repo; }

        [HttpGet]
        public IActionResult GetAll() => Ok(_repo.Jobs.ToList());

        [HttpGet("{id}")]
        public IActionResult Get(int id)
        {
            var job = _repo.GetJobById(id);
            if (job == null) return NotFound();
```

```
                return Ok(job);
            }

            [HttpPost]
            [Authorize(Roles = "Employer")]
            public IActionResult Create(JobDto dto)
            {
                var job = new Job { Title = dto.Title, Company = dto.Company,
Description = dto.Description };
                _repo.CreateJob(job);
                return CreatedAtAction(nameof(Get), new { id = job.Id }, job);
            }

            [HttpPut("{id}")]
            [Authorize(Roles = "Employer")]
            public IActionResult Update(int id, JobDto dto)
            {
                var existing = _repo.GetJobById(id);
                if (existing == null) return NotFound();

                existing.Title = dto.Title;
                existing.Description = dto.Description;
                existing.Company = dto.Company;
                _repo.UpdateJob(existing);
                return NoContent();
            }

            [HttpDelete("{id}")]
            [Authorize(Roles = "Employer")]
            public IActionResult Delete(int id)
            {
                var job = _repo.GetJobById(id);
                if (job == null) return NotFound();
                _repo.DeleteJob(job);
                return NoContent();
            }
        }
    }
```

Лістинг Program.cs:

```
using JobPortal.Api.Data;
using JobPortal.Api.Models;
using JobPortal.Api.Repositories;
using Microsoft.AspNetCore.Authentication.JwtBearer;
using Microsoft.AspNetCore.Identity;
using Microsoft.EntityFrameworkCore;
using Microsoft.IdentityModel.Tokens;
using System.Text;

internal class Program
{
    private static async Task Main(string[] args)
    {
        var builder = WebApplication.CreateBuilder(args);

        // DbContexts
        builder.Services.AddDbContext<PortalDbContext>(opts =>

opts.UseSqlServer(builder.Configuration.GetConnectionString("JobPortalConnection")));

        builder.Services.AddDbContext<AppIdentityDbContext>(opts =>

opts.UseSqlServer(builder.Configuration.GetConnectionString("IdentityConnection")));

        // Identity
```

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

            // Jwt authentication
            var jwt = builder.Configuration.GetSection("Jwt");
            var key = Encoding.UTF8.GetBytes(jwt["Key"]!);

            builder.Services.AddAuthentication(options =>
            {
                options.DefaultAuthenticateScheme =
JwtBearerDefaults.AuthenticationScheme;
                options.DefaultChallengeScheme = JwtBearerDefaults.AuthenticationScheme;
            })
            .AddJwtBearer(options =>
            {
                options.RequireHttpsMetadata = false;
                options.SaveToken = true;
                options.TokenValidationParameters = new TokenValidationParameters
                {
                    ValidateIssuer = true,
                    ValidateAudience = true,
                    ValidateLifetime = true,
                    ValidateIssuerSigningKey = true,
                    ValidIssuer = jwt["Issuer"],
                    ValidAudience = jwt["Audience"],
                    IssuerSigningKey = new SymmetricSecurityKey(key)
                };
            });

            builder.Services.AddScoped<IPortalRepository, EFPortalRepository>();

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

            var app = builder.Build();

            using (var scope = app.Services.CreateScope())
            {
                var services = scope.ServiceProvider;

                var identityContext =
services.GetRequiredService<AppIdentityDbContext>();
                var portalContext = services.GetRequiredService<PortalDbContext>();

                try
                {
                    // AUTOMATICALLY APPLY MIGRATIONS / CREATE DB
                    await identityContext.Database.MigrateAsync();
                    await portalContext.Database.MigrateAsync();

                    var roleManager =
services.GetRequiredService<RoleManager<IdentityRole>>();
                    string[] roles = { "Candidate", "Employer" };
                    foreach (var r in roles)
                    {
                        if (!await roleManager.RoleExistsAsync(r))
                        {
                            await roleManager.CreateAsync(new IdentityRole(r));
                        }
```

```
                }
            }
            catch (Exception ex)
            {
                var logger = services.GetRequiredService<ILogger<Program>>();
                logger.LogError(ex, "An error occurred while migrating the
database.");
            }
        }

        app.UseAuthentication();
        app.UseAuthorization();

        app.MapControllers();

        app.Run();
    }
}
```

Результат роботи проекту:

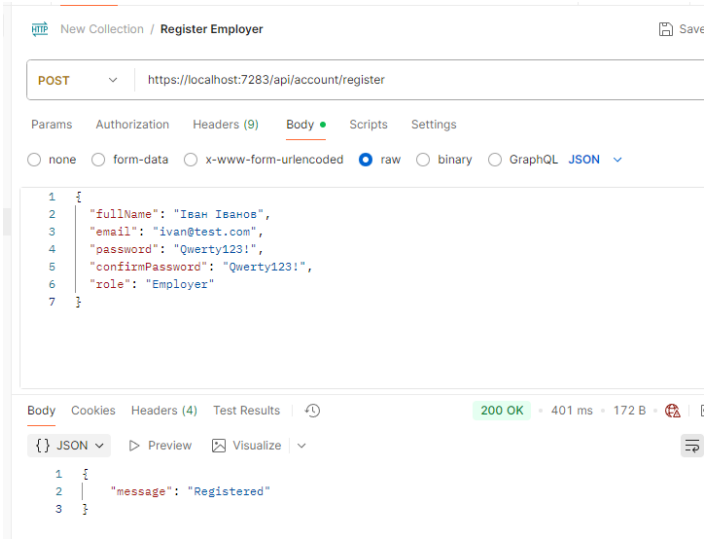

Рис. 2.2. Реєстрація нового користувача з правами Роботодавця

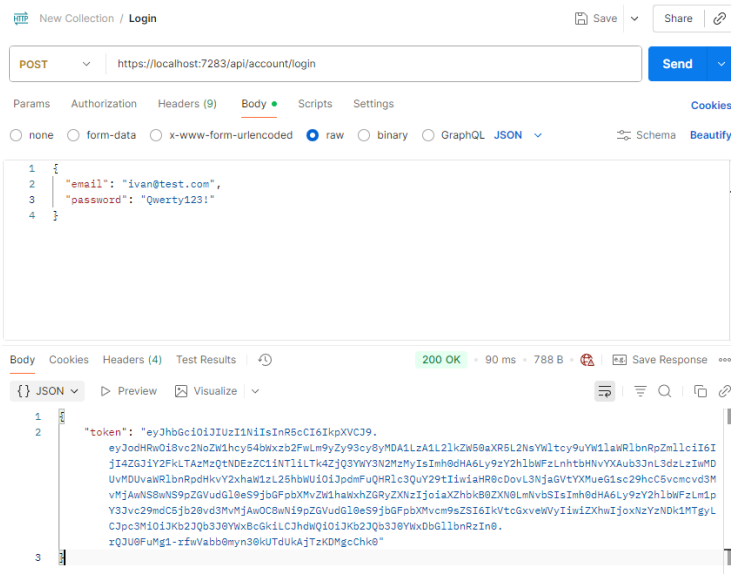

Рис. 2.3. Вхід у аккаунт користувача-роботодавця

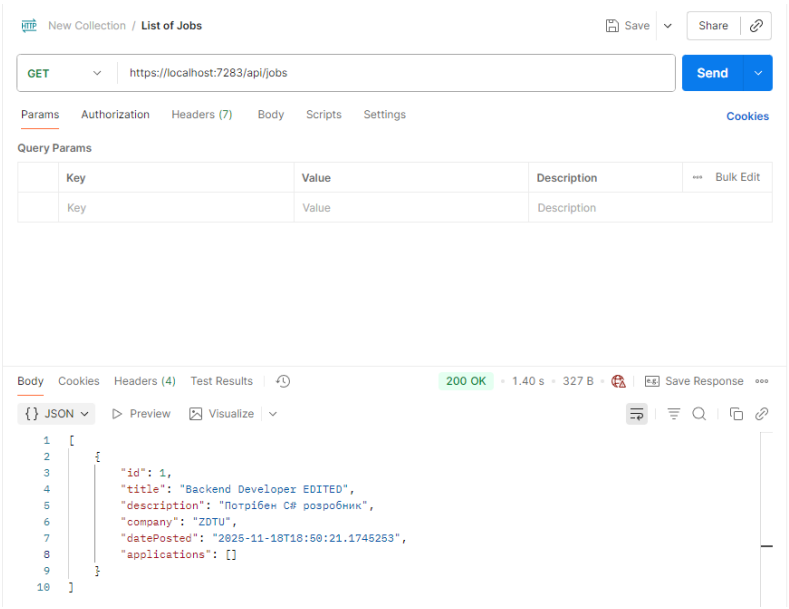

Рис. 2.4. Список всіх наявних вакансій

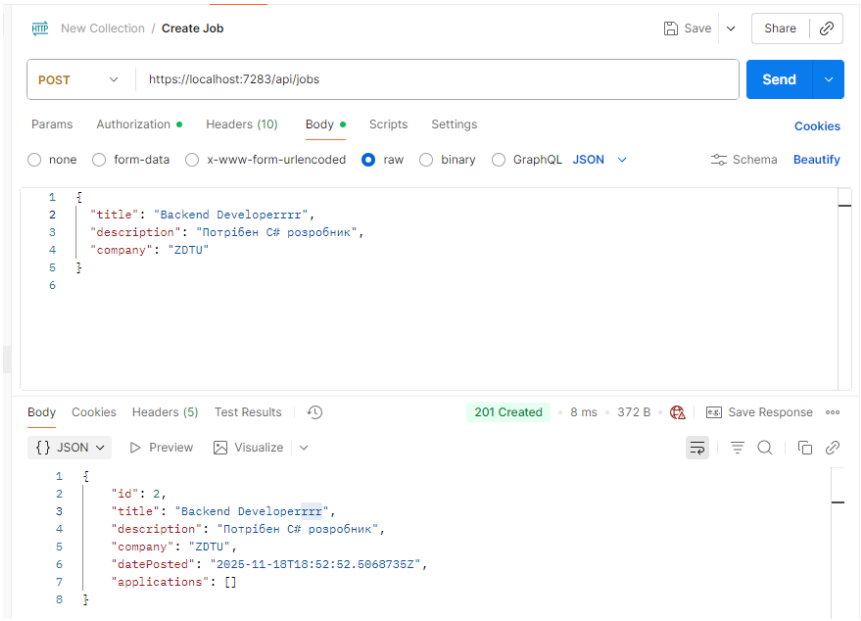

Рис. 2.5. Створення нової вакансії (тільки для роботодавця)

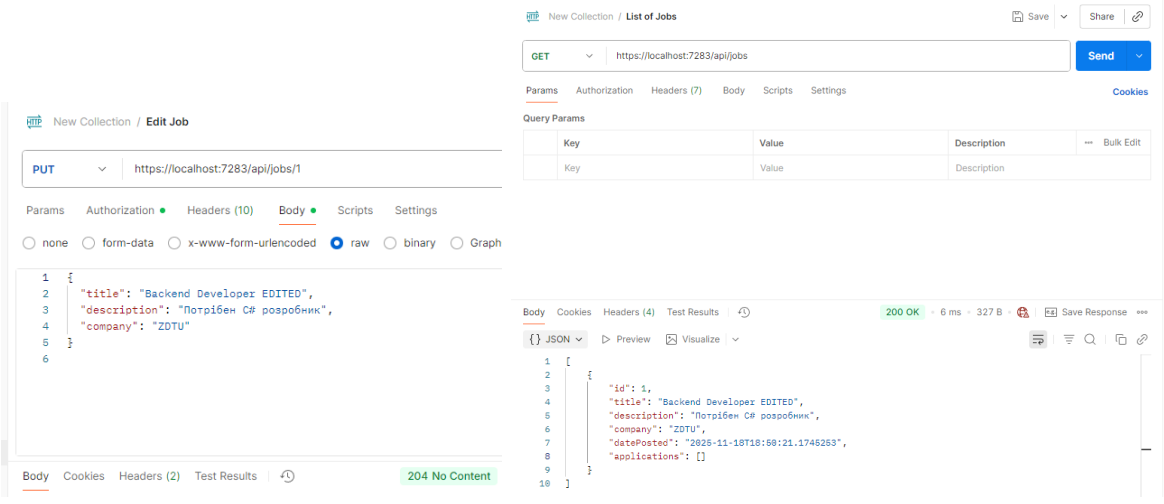

Рис. 2.6. Редагування існуючої вакансії (тільки для роботодавця)

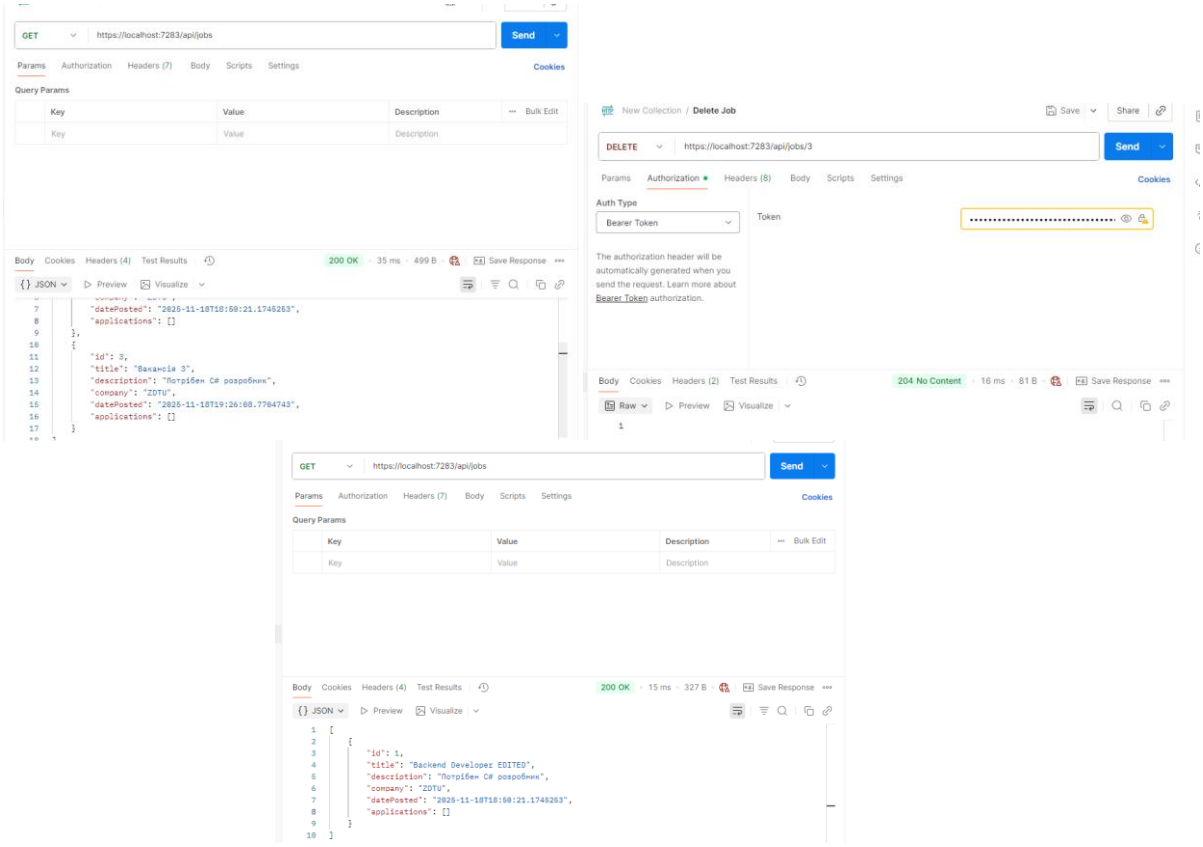

Рис. 2.7. Видалення вакансії (тільки для роботодавця)

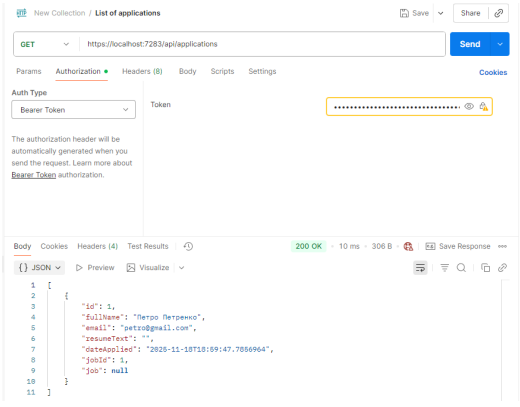

Рис. 2.8. Список відгуків на певну вакансію

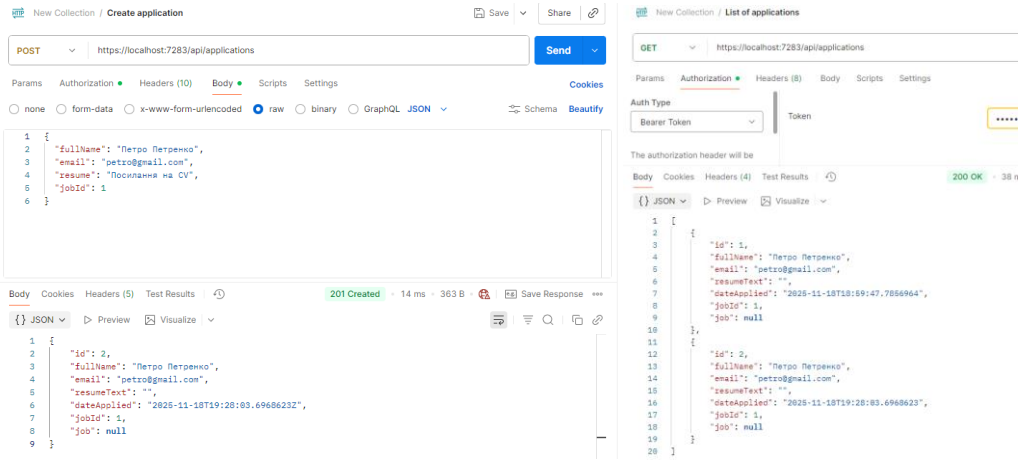

Рис. 2.9. Створення нового відгуку

Завдання 3.

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

1. Клієнт відправляє запит на login-ендпоїнт зі своїм логіном та паролем.
2. Якщо користувач з таким логіном та паролем існує, тоді сервер повинен повернути згенерований JSON Web Token.

Лістинг:

```
        [HttpPost("login")]
        public async Task<IActionResult> Login(LoginDto dto)
        {
            var user = await _userManager.FindByEmailAsync(dto.Email);
            if (user == null) return Unauthorized();

            var chk = await _signInManager.CheckPasswordSignInAsync(user,
dto.Password, false);
            if (!chk.Succeeded) return Unauthorized();

            var roles = await _userManager.GetRolesAsync(user);

            var claims = new List<Claim>
            {
                new Claim(ClaimTypes.NameIdentifier, user.Id),
                new Claim(ClaimTypes.Name, user.UserName ?? user.Email ?? ""),
                new Claim(ClaimTypes.Email, user.Email ?? "")
            };
            foreach (var r in roles) claims.Add(new Claim(ClaimTypes.Role, r));

            var key = new
SymmetricSecurityKey(Encoding.UTF8.GetBytes(_config["Jwt:Key"]));
            var creds = new SigningCredentials(key,
SecurityAlgorithms.HmacSha256);
            var token = new JwtSecurityToken(
                issuer: _config["Jwt:Issuer"],
                audience: _config["Jwt:Audience"],
                claims: claims,
                expires:
DateTime.UtcNow.AddMinutes(double.Parse(_config["Jwt:DurationMinutes"])),
                signingCredentials: creds
            );

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

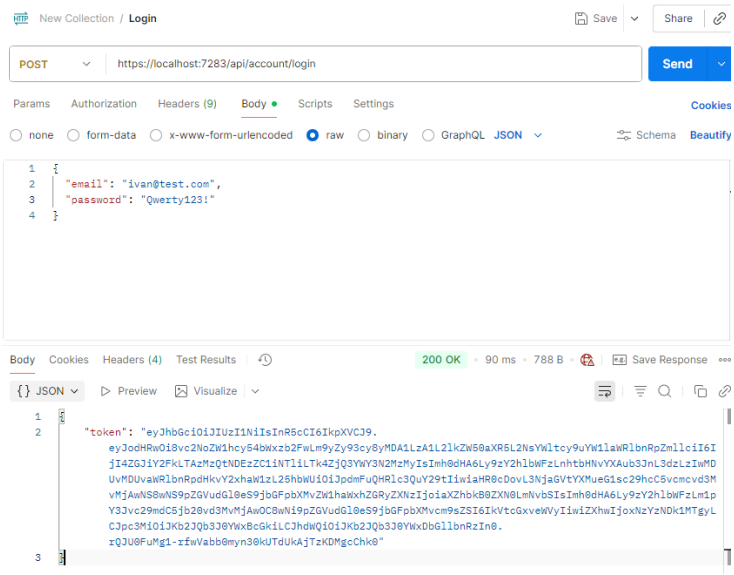

Рис. 3.1. login-ендпоїнт

3. 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.

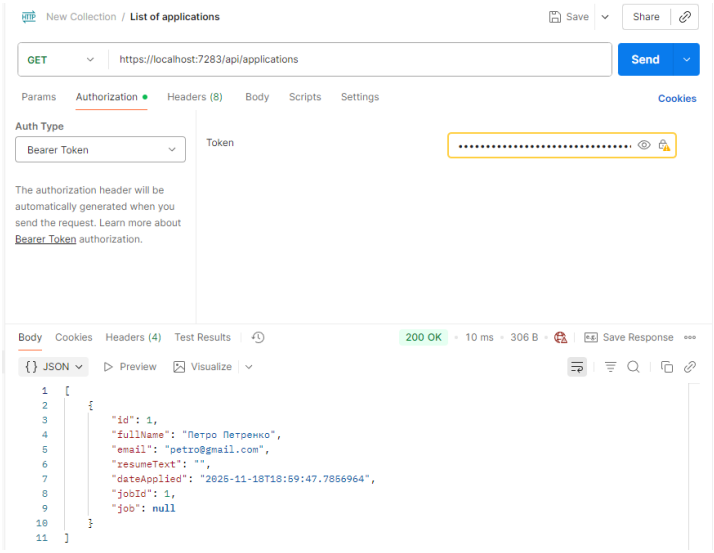

Рис. 3.2. заголовок Authorization зі значенням Bearer TokenValue

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

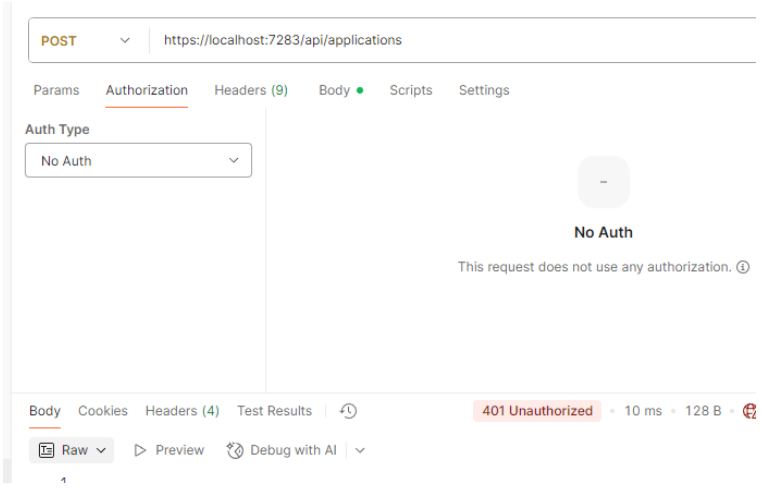

Рис. 3.3. помилка 401 Unauthorized, якщо не вказано токен

Висновок: в рамках цієї лабораторної роботи були успішно набуті ключові навички роботи з сучасною архітектурою веб-додатків, сфокусовані на розробці Web API та реалізації механізмів безпеки в середовищі ASP.NET Core.