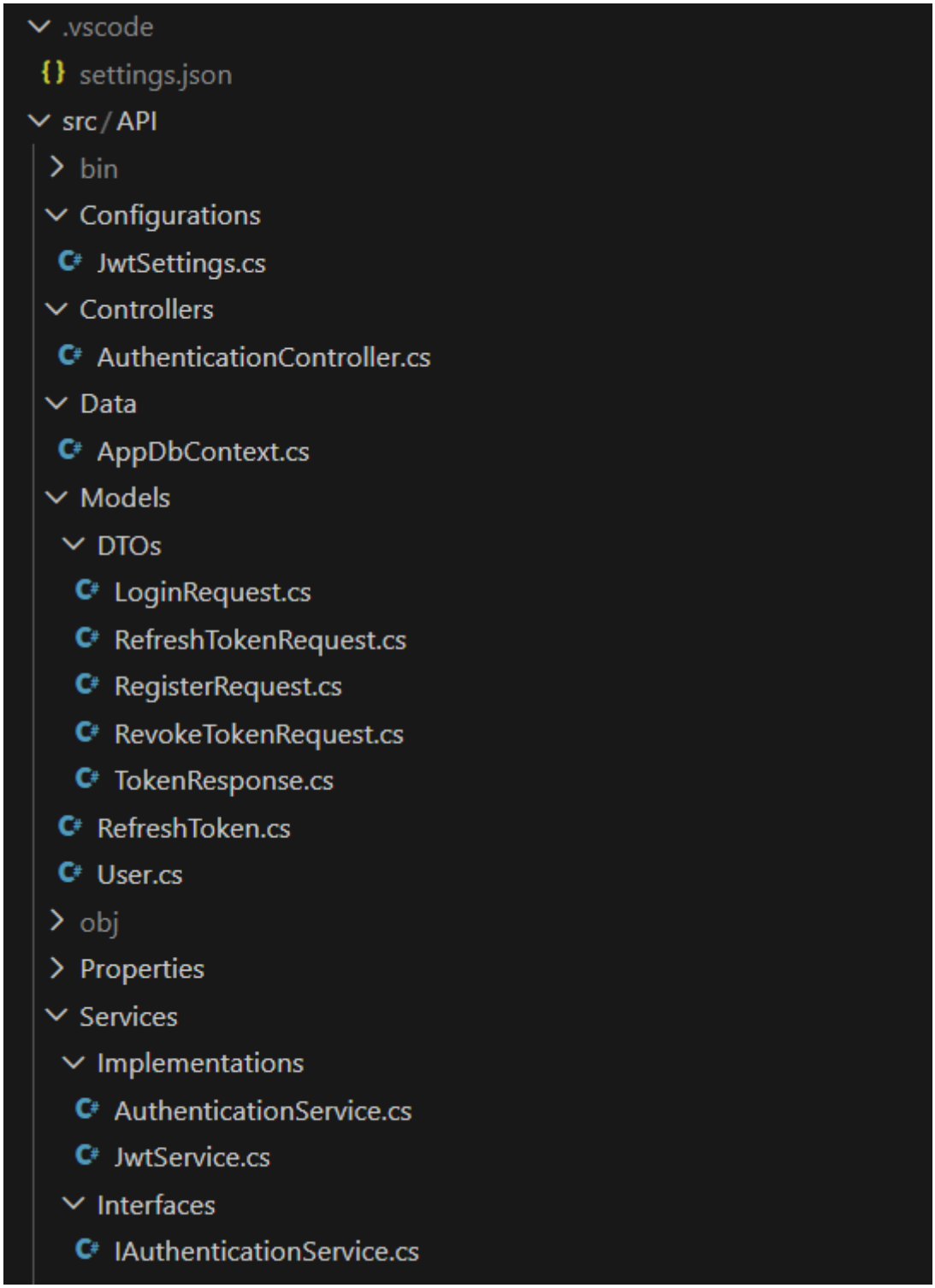
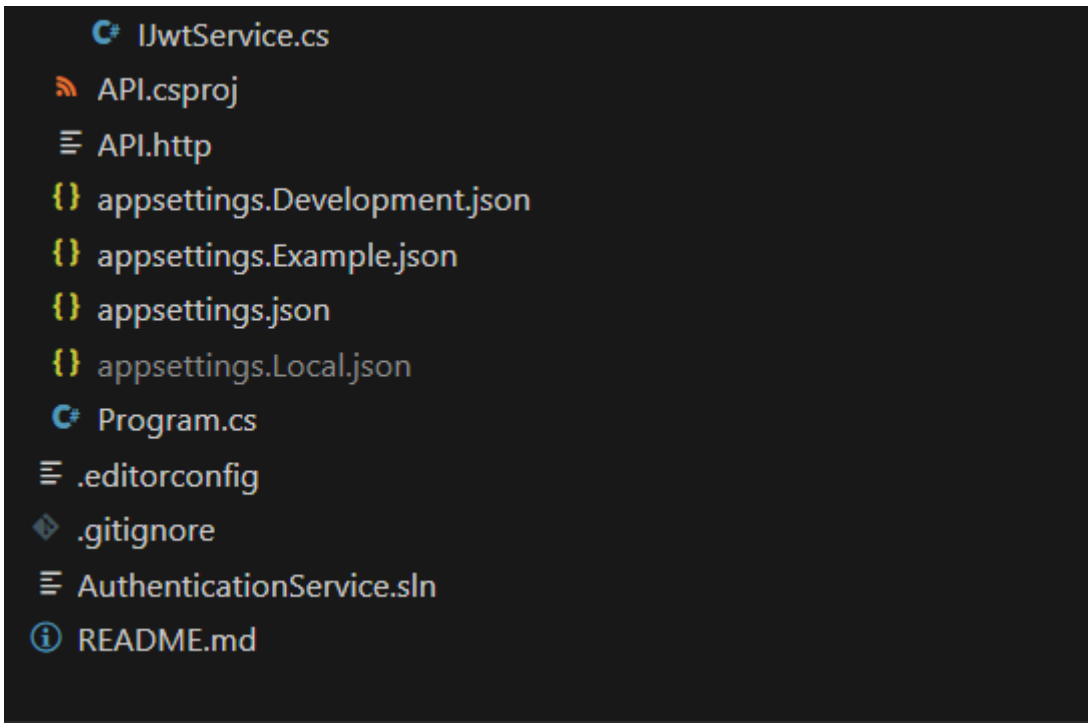
Отчет по выполнению Домашнего Задания

Никандров Валентин ИУ5-35Б

Файловая структура проекта:





Листинг кода проекта:

```
using API.Configurations;
using API.Data;
using API.Services;
using System.Text;
using Microsoft.AspNetCore.Authentication.JwtBearer;
using Microsoft.EntityFrameworkCore;
using Microsoft.IdentityModel.Tokens;

var builder = WebApplication.CreateBuilder(args);

builder.Configuration
    .AddJsonFile("appsettings.json", optional: false)
    .AddJsonFile("appsettings.Development.json", optional: true)
    .AddJsonFile("appsettings.Local.json", optional: true)
    .AddEnvironmentVariables();

var configuration = builder.Configuration;

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

builder.Services.AddDbContext<AppDbContext>(options =>
options.UseNpgsql(configuration.GetConnectionString("DefaultConnection")));

var jwtSettings = configuration.GetSection("JwtSettings").Get<JwtSettings>();
```

```
if (jwtSettings is null) throw new InvalidOperationException("JwtSettings
configuration is missing");

builder.Services.AddSingleton(jwtSettings);

builder.Services.AddScoped<IJwtService, JwtService>();
builder.Services.AddScoped<IAuthenticationService, AuthenticationService>();

builder.Services.AddAuthentication(options =>
{
    options.DefaultAuthenticateScheme = JwtBearerDefaults.AuthenticationScheme;
    options.DefaultChallengeScheme = JwtBearerDefaults.AuthenticationScheme;
}).AddJwtBearer(options =>
{
    options.TokenValidationParameters = new TokenValidationParameters
    {
        ValidateIssuer = true,
        ValidateAudience = true,
        ValidateLifetime = true,
        ValidateIssuerSigningKey = true,
        ValidIssuer = jwtSettings.Issuer,
        ValidAudience = jwtSettings.Audience,
        IssuerSigningKey = new
SymmetricSecurityKey(Encoding.UTF8.GetBytes(jwtSettings.SecretKey)),
        ClockSkew = TimeSpan.Zero
    };

    options.Events = new JwtBearerEvents
    {
        OnAuthenticationFailed = context =>
        {
            if (context.Exception.GetType() ==
typeof(SecurityTokenExpiredException))
            {
                context.Response.Headers["Token-Expired"] = "true";
            }

            return Task.CompletedTask;
        }
    };
});

builder.Services.AddAuthorization();

var app = builder.Build();

if (app.Environment.IsDevelopment())
{
    app.UseSwagger();
    app.UseSwaggerUI();
}
```

```
app.UseHttpsRedirection();
app.UseAuthentication();
app.UseAuthorization();
app.MapControllers();

using (var scope = app.Services.CreateScope())
{
    var dbContext = scope.ServiceProvider.GetRequiredService<AppDbContext>();

    // if (!dbContext.Users.Any())
    // {
    //     var authService =
scope.ServiceProvider.GetRequiredService<IAuthenticationService>();
    //     await authService.RegisterAsync("Ivan", "Ivanov", "test@example.com",
"qwerty123123");
    //     Console.WriteLine("Test user created: test@example.com/
qwerty123123");
    // }
}

app.Run();
```

```
namespace API.Configurations
{
    public class JwtSettings
    {
        public required string SecretKey { get; set; }
        public required string Issuer { get; set; }
        public required string Audience { get; set; }
        public required int AccessTokenExpirationMinutes { get; set; }
        public required int RefreshTokenExpirationDays { get; set; }
    }
}
```

```
using System.ComponentModel.DataAnnotations.Schema;

namespace API.Models
{
    [Table("users")]
    public class User
    {
        [Column("id")]
        public int Id { get; set; }
        [Column("name")]
        public required string Name { get; set; }
        [Column("surname")]
```

```
        public required string Surname { get; set; }
        [Column("email")]
        public required string Email { get; set; }
        [Column("password_hash")]
        public required string PasswordHash { get; set; }
        [Column("created_at")]
        public DateTime CreatedAt { get; set; }
        [Column("updated_at")]
        public DateTime UpdatedAt { get; set; }
    };
}
```

```
using System.ComponentModel.DataAnnotations.Schema;

namespace API.Models
{
    [Table("refresh_tokens")]
    public class RefreshToken
    {
        [Column("id")]
        public int Id { get; set; }
        [Column("user_id")]
        public required int UserId { get; set; }
        [ForeignKey(nameof(UserId))]
        public virtual User? User { get; set; }
        [Column("token")]
        public required string Token { get; set; }
        [Column("expires_at")]
        public required DateTime ExpiresAt { get; set; }
        [Column("created_at")]
        public DateTime CreatedAt { get; set; }
        [Column("updated_at")]
        public DateTime UpdatedAt { get; set; }
        [NotMapped]
        public bool isExpired => DateTime.UtcNow >= ExpiresAt;
    }
}
```

```
namespace API.Models.Responses
{
    public class TokenResponse
    {
        public required string AccessToken { get; set; }
        public required string RefreshToken { get; set; }
        public required DateTime ExpiresAt { get; set; }
    }
}
```

```
namespace API.Models.Requests
{
    public class RevokeTokenRequest
    {
        public required string RefreshToken { get; set; }
    }
}
```

```
namespace API.Models.Requests
{
    public class RegisterRequest
    {
        public required string Name { get; set; }
        public required string Surname { get; set; }
        public required string Email { get; set; }
        public required string Password { get; set; }
    }
}
```

```
namespace API.Models.Requests
{
    public class RefreshTokenRequest
    {
        public required string AccessToken { get; set; }
        public required string RefreshToken { get; set; }
    }
}
```

```
namespace API.Models.Requests
{
    public class LoginRequest
    {
        public required string Email { get; set; }
        public required string Password { get; set; }
    }
}
```

```
using API.Models;
using Microsoft.EntityFrameworkCore;

namespace API.Data
```

```
{
    public class AppDbContext : DbContext
    {
        public AppDbContext(DbContextOptions<AppDbContext> options) :
base(options)
        {

        }

        public DbSet<User> Users { get; set; }
        public DbSet<RefreshToken> RefreshTokens { get; set; }

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

            modelBuilder.Entity<User>(entity =>
            {
                entity.HasKey(e => e.Id);
                entity.Property(e => e.Name).IsRequired().HasMaxLength(64);
                entity.Property(e => e.Surname).IsRequired().HasMaxLength(64);
                entity.Property(e => e.Email).IsRequired().HasMaxLength(100);
                entity.Property(e => e.CreatedAt).IsRequired();
                entity.Property(e => e.UpdatedAt);
            });

            modelBuilder.Entity<RefreshToken>(entity =>
            {
                entity.HasKey(e => e.Id);
                entity.Property(e => e.UserId).IsRequired();
                entity.Property(e => e.Token).IsRequired();
                entity.Property(e => e.ExpiresAt).IsRequired();
                entity.Property(e => e.CreatedAt).IsRequired();

                entity.HasOne(rt => rt.User).WithMany().HasForeignKey(rt =>
rt.UserId).OnDelete(DeleteBehavior.Cascade);
            });
        }
    }
}
```

```
using API.Models;
using System.Security.Claims;

namespace API.Services
{
    public interface IJwtService
    {
        string GenerateAccessToken(User user);
        string GenerateRefreshToken();
```

```
        ClaimsPrincipal GetPrincipalFromExpiredToken(string token);
    }
}
```

```
using API.Models;
using API.Models.Responses;

namespace API.Services
{
    public interface IAuthenticationService
    {
        Task<User> RegisterAsync(string name, string surname, string email,
string password);
        Task<User?> AuthenticateAsync(string email, string password);
        Task<TokenResponse> GenerateTokensAsync(User user);
        Task<TokenResponse> RefreshTokenAsync(string accessToken, string
refreshToken);
        Task RevokeRefreshTokenAsync(string refreshToken);
    }
}
```

```
using System.IdentityModel.Tokens.Jwt;
using System.Security.Claims;
using System.Security.Cryptography;
using System.Text;
using API.Models;
using API.Configurations;
using Microsoft.IdentityModel.Tokens;

namespace API.Services
{
    public class JwtService : IJwtService
    {
        private readonly JwtSettings _jwtSettings;

        public JwtService(JwtSettings jwtSettings)
        {
            _jwtSettings = jwtSettings;
        }

        public string GenerateAccessToken(User user)
        {
            var tokenHandler = new JwtSecurityTokenHandler();
            var key = Encoding.UTF8.GetBytes(_jwtSettings.SecretKey);

            var claims = new List<Claim>
            {
```

```
            new Claim(ClaimTypes.NameIdentifier, user.Id.ToString()),
            new Claim(ClaimTypes.Name, user.Name),
            new Claim(ClaimTypes.Surname, user.Surname),
            new Claim(ClaimTypes.Email, user.Email),
            new Claim(JwtRegisteredClaimNames.Jti, Guid.NewGuid().ToString())
        };

        var tokenDescriptor = new SecurityTokenDescriptor
        {
            Subject = new ClaimsIdentity(claims),
            Expires =
DateTime.UtcNow.AddMinutes(_jwtSettings.AccessTokenExpirationMinutes),
            Issuer = _jwtSettings.Issuer,
            Audience = _jwtSettings.Audience,
            SigningCredentials = new SigningCredentials(new
SymmetricSecurityKey(key), SecurityAlgorithms.HmacSha256Signature)
        };

        var token = tokenHandler.CreateToken(tokenDescriptor);
        return tokenHandler.WriteToken(token);
    }

    public string GenerateRefreshToken()
    {
        var randomNumber = new byte[32];
        using var randomNumberGenerator = RandomNumberGenerator.Create();

        randomNumberGenerator.GetBytes(randomNumber);

        return Convert.ToBase64String(randomNumber);
    }

    public ClaimsPrincipal GetPrincipalFromExpiredToken(string token)
    {
        var tokenValidationParameters = new TokenValidationParameters
        {
            ValidateAudience = false,
            ValidateIssuer = false,
            IssuerSigningKey = new
SymmetricSecurityKey(Encoding.UTF8.GetBytes(_jwtSettings.SecretKey)),
            ValidateLifetime = false
        };

        var tokenHandler = new JwtSecurityTokenHandler();
        var principal = tokenHandler.ValidateToken(token,
tokenValidationParameters, out var securityToken);

        if (securityToken is not JwtSecurityToken jwtSecurityToken ||
!jwtSecurityToken.Header.Alg.Equals(SecurityAlgorithms.HmacSha256,
StringComparison.InvariantCultureIgnoreCase))
        {
```

```
                throw new SecurityTokenException("SecurityTokenException: Invalid
token");
            }

            return principal;
        }
    }
}
```

```
using API.Data;
using API.Models;
using API.Models.Responses;
using Microsoft.EntityFrameworkCore;
using Microsoft.IdentityModel.Tokens;
using System.Security.Claims;
using BCrypt.Net;
using System.Text;

namespace API.Services
{
    public class AuthenticationService : IAuthenticationService
    {
        private readonly AppDbContext _context;
        private readonly IJwtService _jwtService;
        private readonly IConfiguration _configuration;

        public AuthenticationService(AppDbContext context, IJwtService
generationService, IConfiguration configuration)
        {
            _context = context;
            _jwtService = generationService;
            _configuration = configuration;
        }

        public async Task<User> RegisterAsync(string name, string surname, string
email, string password)
        {
            if (await _context.Users.AnyAsync(u => u.Email == email)) throw new
Exception("Email already exists");

            var user = new User
            {
                Name = name,
                Surname = surname,
                Email = email,
                PasswordHash = HashPassword(password),
                CreatedAt = DateTime.UtcNow
            };
```

```
            _context.Add(user);
            await _context.SaveChangesAsync();

            return user;
        }

        public async Task<User?> AuthenticateAsync(string email, string password)
        {
            var user = await _context.Users.FirstOrDefaultAsync(u => u.Email ==
email);

            if (user is null || !VerifyPassword(password, user.PasswordHash))
return null;

            return user;
        }

        public async Task<TokenResponse> GenerateTokensAsync(User user)
        {
            var accessToken = _jwtService.GenerateAccessToken(user);
            var refreshToken = _jwtService.GenerateRefreshToken();

            var refreshTokenEntity = new RefreshToken
            {
                UserId = user.Id,
                Token = refreshToken,
                ExpiresAt =
DateTime.UtcNow.AddDays(_configuration.GetValue<int>("JwtSettings:RefreshTokenExp
irationDays")),
                CreatedAt = DateTime.UtcNow
            };

            _context.RefreshTokens.Add(refreshTokenEntity);
            await _context.SaveChangesAsync();

            return new TokenResponse
            {
                AccessToken = accessToken,
                RefreshToken = refreshToken,
                ExpiresAt =
DateTime.UtcNow.AddMinutes(_configuration.GetValue<int>("JwtSettings::AccessToken
ExpirationMinutes"))
            };
        }

        public async Task<TokenResponse> RefreshTokenAsync(string accessToken,
string refreshToken)
        {
            var principal =
_jwtService.GetPrincipalFromExpiredToken(accessToken);
```

```
            var userId =
int.Parse(principal.FindFirst(ClaimTypes.NameIdentifier)?.Value ?? throw new
SecurityTokenException("Invalid token"));

            var storedRefreshToken = await _context.RefreshTokens.Include(rt =>
rt.User).FirstOrDefaultAsync(rt => rt.Token == refreshToken && rt.UserId ==
userId);


            if (storedRefreshToken is null || storedRefreshToken.User is null ||
storedRefreshToken.isExpired) throw new SecurityTokenException("Invalid refresh
token");

            var newAccessToken =
_jwtService.GenerateAccessToken(storedRefreshToken.User);
            var newRefreshToken = _jwtService.GenerateRefreshToken();

            _context.Remove(storedRefreshToken);

            var newRefreshTokenEntity = new RefreshToken
            {
                UserId = storedRefreshToken.UserId,
                User = storedRefreshToken.User,
                Token = newRefreshToken,
                ExpiresAt =
DateTime.UtcNow.AddDays(_configuration.GetValue<int>("JwtSettings:RefreshTokenExp
irationDays")),
                CreatedAt = DateTime.UtcNow
            };

            _context.Add(newRefreshTokenEntity);
            await _context.SaveChangesAsync();

            return new TokenResponse
            {
                AccessToken = newAccessToken,
                RefreshToken = newRefreshToken,
                ExpiresAt =
DateTime.UtcNow.AddMinutes(_configuration.GetValue<int>("JwtSettings:AccessTokenE
xpirationMinutes"))
            };
        }

        public async Task RevokeRefreshTokenAsync(string refreshToken)
        {
            var token = await _context.RefreshTokens.FirstOrDefaultAsync(rt =>
rt.Token == refreshToken);

            if (token != null)
            {
                _context.RefreshTokens.Remove(token);
```

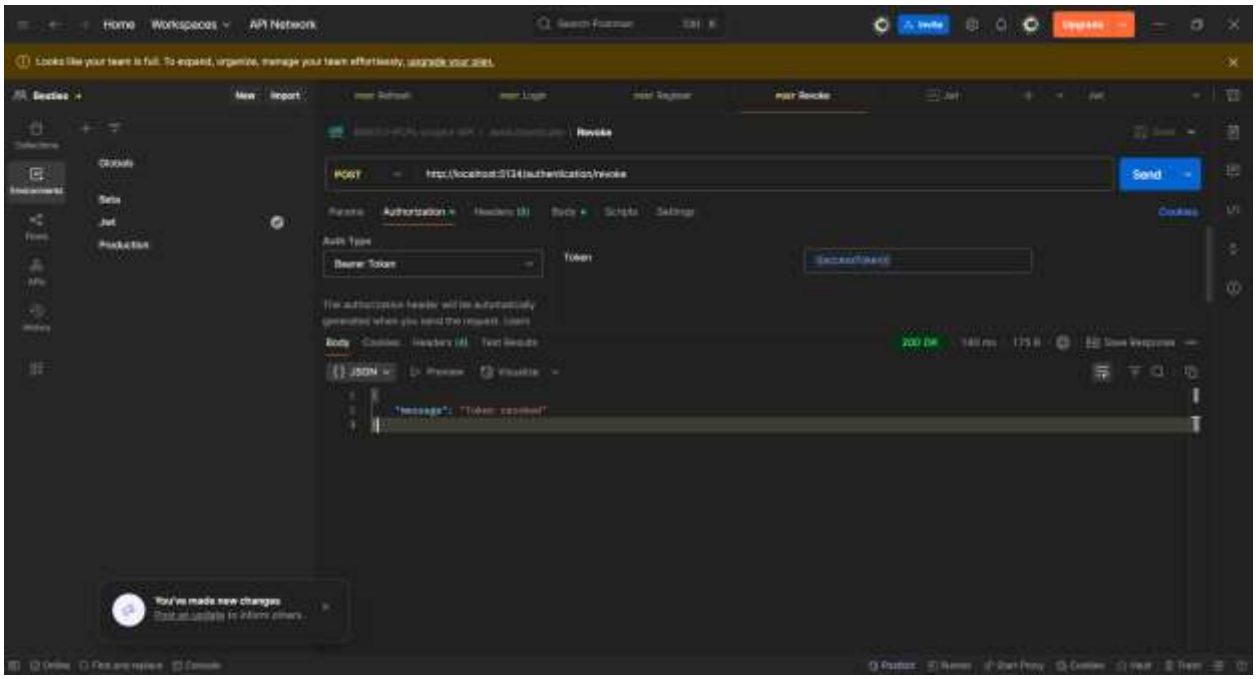
```
                await _context.SaveChangesAsync();
            }
        }

        private string HashPassword(string password)
        {
            return BCrypt.Net.BCrypt.EnhancedHashPassword(password,
HashType.SHA256);
        }

        private bool VerifyPassword(string password, string storedHash)
        {
            try
            {
                return BCrypt.Net.BCrypt.EnhancedVerify(password, storedHash,
HashType.SHA256);
            }
            catch (SaltParseException)
            {
                return false;
            }
        }
    }
}
```

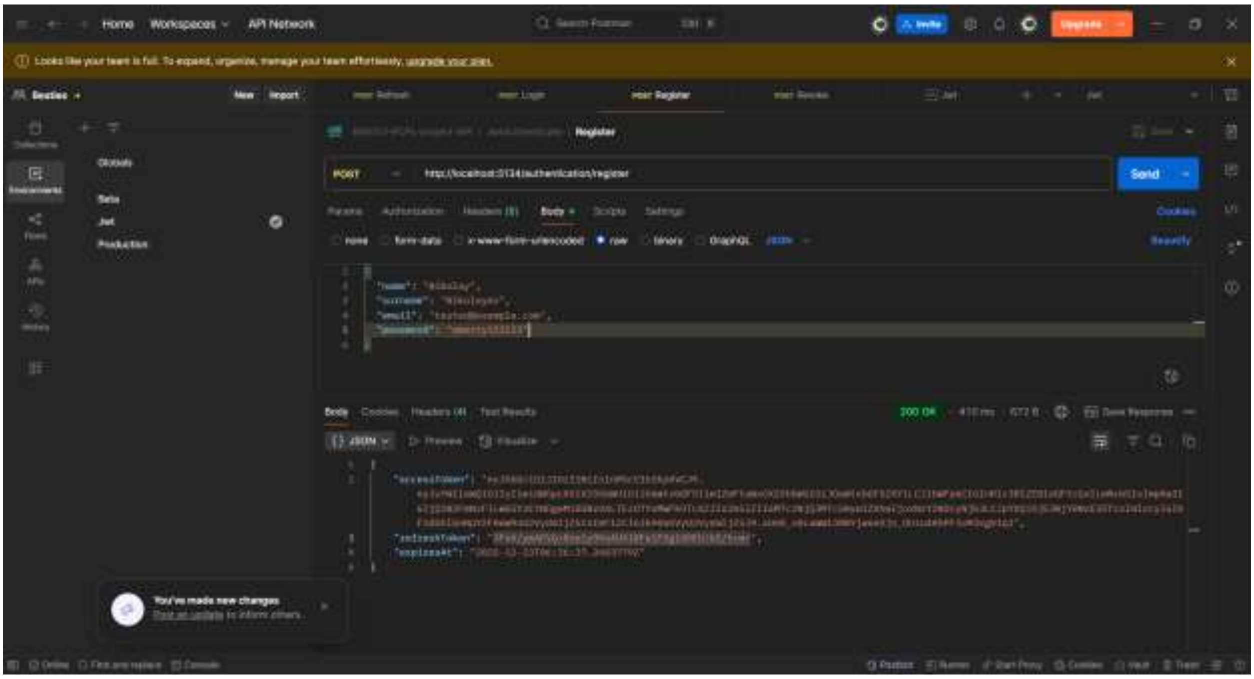
```
using API.Services;
using API.Models.Requests;
using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.Mvc;
using Microsoft.IdentityModel.Tokens;
using Microsoft.AspNetCore.Authorization;

namespace API.Controllers
{
    [Route("[controller]")]
    [ApiController]
    public class AuthenticationController : ControllerBase
    {
        private readonly IAuthenticationService _authenticationService;

        public AuthenticationController(IAuthenticationService
authenticationService)
        {
            _authenticationService = authenticationService;
        }

        [HttpPost("register")]
        public async Task<IActionResult> Register([FromBody] RegisterRequest
request)
```

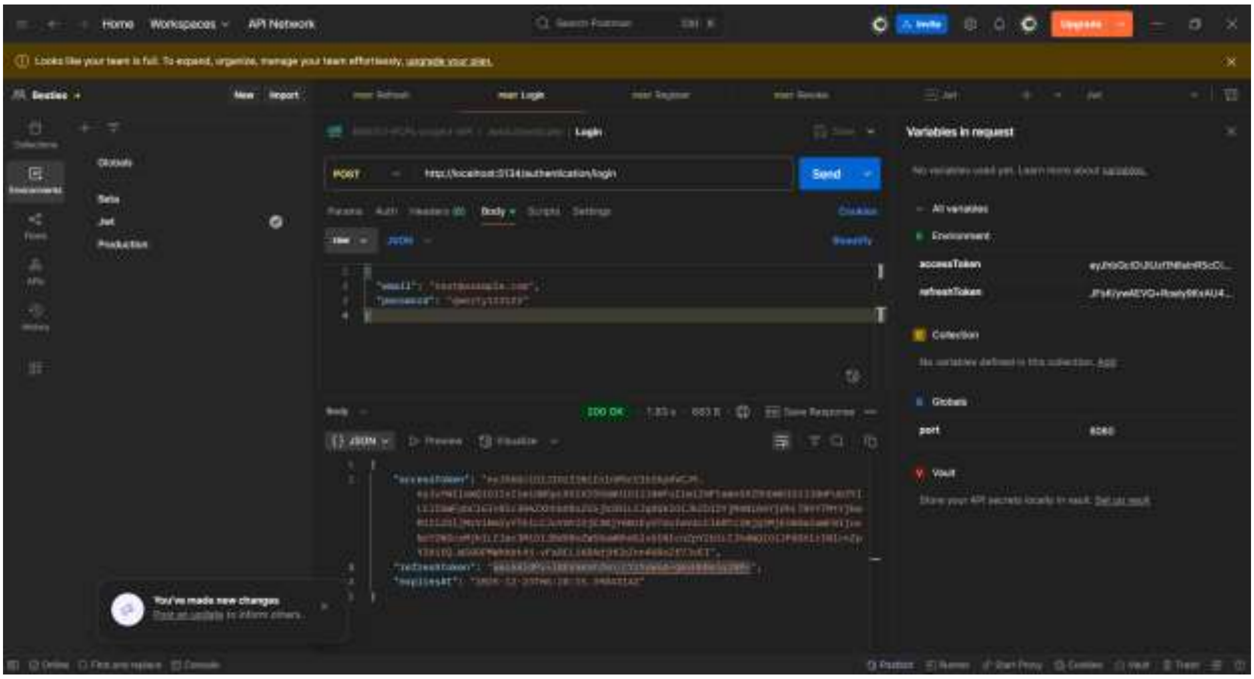
```
        {
            try
            {
                var user = await
_authenticationService.RegisterAsync(request.Name, request.Surname,
request.Email, request.Password);
                var tokens = await
_authenticationService.GenerateTokensAsync(user);
                return Ok(tokens);
            }
            catch (Exception exception)
            {
                return BadRequest(new { message = exception.Message });
            }
        }

        [HttpPost("login")]
        public async Task<IActionResult> Login([FromBody] LoginRequest request)
        {
            var user = await
_authenticationService.AuthenticateAsync(request.Email, request.Password);

            if (user is null) return Unauthorized(new { message = "Invalid
credentials" });

            var tokens = await _authenticationService.GenerateTokensAsync(user);
            return Ok(tokens);
        }

        [HttpPost("refresh")]
        public async Task<IActionResult> Refresh([FromBody] RefreshTokenRequest
request)
        {
            try
            {
                var tokens = await
_authenticationService.RefreshTokenAsync(request.AccessToken,
request.RefreshToken);
                return Ok(tokens);
            }
            catch (SecurityTokenException exception)
            {
                return Unauthorized(new { message = exception.Message });
            }
        }

        [HttpPost("revoke")]
        [Authorize]
        public async Task<IActionResult> Revoke([FromBody] RevokeTokenRequest
request)
        {
```

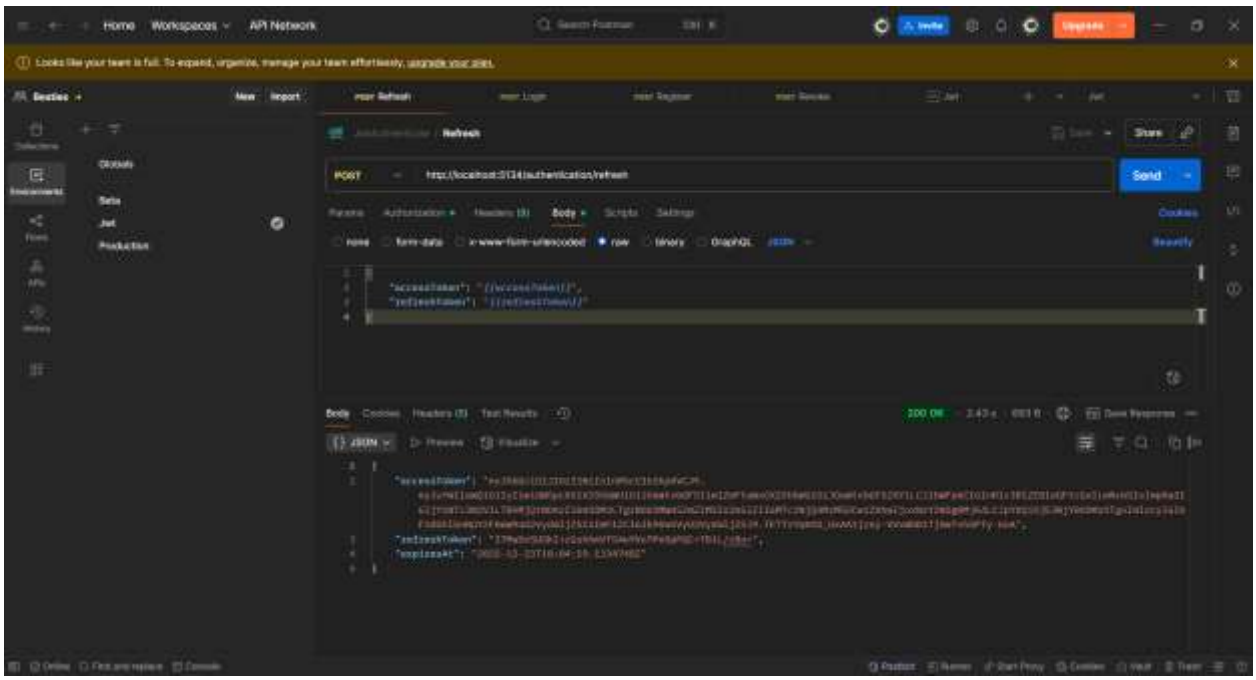
```
            await
_authenticationService.RevokeRefreshTokenAsync(request.RefreshToken);

            return Ok(new { message = "Token revoked" });
        }
    }
}
```

Результат работы программы:




Home
Workspaces
API Network
Register
POST
Send
Params
Body
Cookies
Beautify
200 OK
You've made new changes


Home
Workspaces
API Network
Login
POST
Send
Body
Cookies
Beautify
200 OK
Variables in request
All variables
Environment
accessToken
refreshToken
Collection
Globals
port
Vault
You've made new changes


Home
Workspaces
API Network
Looks like your team is full. To expand, organize, manage your team effortlessly, upgrade your plan.
Invite
Upgrade
Share
Refresh
POST
http://localhost:3134/authentication/refresh
Send
Params
Authorization
Headers (8)
Body
Scripts
Settings
Cookies
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
JSON
Beautify
{
"accessToken": "{{accessToken}}",
"refreshToken": "{{refreshToken}}"
}
Body
Cookies
Headers (7)
Test Results
200 OK
JSON
Preview
Visualize
{
"accessToken":
"refreshToken":
"expiresAt":
}