Sprint 4. JWT. Реализация сервиса

- создайте интерфейс

```
    public interface ITokenService
    {
        string CreateToken(int UserId);
    }
```

- установите пакет `Microsoft.IdentityModel.Tokens` и `System.IdentityModel.Tokens.Jwt`:

```
    <PackageReference Include="Microsoft.IdentityModel.Tokens" Version="7.7.1" />
    <PackageReference Include="System.IdentityModel.Tokens.Jwt" Version="7.7.1"/>
```

- реализуйте сервис для генерации jwt - токена в папке `Services`.

```
public class TokenService : ITokenService
{

    private readonly SymmetricSecurityKey _key;
    public TokenService(IConfiguration config)
    {
      _key = new
SymmetricSecurityKey(Encoding.UTF8.GetBytes(config["TokenKey"]!));
    }

    public string CreateToken(int UserId)
    {
       var claims =  new List<Claim>{
         new Claim(JwtRegisteredClaimNames.Name, UserId.ToString())
       };

       var creds = new SigningCredentials(_key,
SecurityAlgorithms.HmacSha512Signature);

       var tokenDecriptor = new SecurityTokenDescriptor(){
         Subject = new ClaimsIdentity(claims),
         Expires = DateTime.UtcNow.AddDays(7),
         SigningCredentials = creds
       };

       var tokenHandler = new JwtSecurityTokenHandler();
       var token = tokenHandler.CreateToken(tokenDecriptor);
       return tokenHandler.WriteToken(token);
```

```
    }
}
```

Примечание: значение `config["TokenKey"]` мы получаем из конфигурации файла `appsettings.Development.json`. Это открытый ключ шифрования. Добавьте данную настройку после после настроек логгирования.

appsettings.Development.json

```
"TokenKey": "super key password for jwt token token token token token token token token token "
```

Изменение модели данных User

- в модель `User` добавьте новое свойство `public string Token {get; set;}`.
- создайте миграцию и примените для обновления базы данных.
- зарегистрируйте в контейнер DI сервис `TokenService`.
- внедрите объект `ITokenService` в конструктор `UserController`.
- в методе создания пользователя измените входную модель с `User` на `UserDto`
- в методе создания пользователя сформируйте из данных модели представления объект `User`, сгенерировав `PasswordHash` и `PasswordSalt`
- примените метод генерации токена из сервиса к полю `Token` у пользователя.
- проверьте конечную точку создания пользователя: у пользователя должны быть хэши паролей и токен
- измените валидаторы с проверки поля `Name` на проверку поля `Login`.

Middleware

- установите пакет `Microsoft.AspNetCore.Authentication.JwtBearer`

Program.cs

```
builder.Services.AddAuthentication(JwtBearerDefaults.AuthenticationScheme)
    .AddJwtBearer(options =>
    {
        options.SaveToken = true;
        options.RequireHttpsMetadata = false;
        options.TokenValidationParameters = new TokenValidationParameters
        {
            ValidateIssuer = false,
            ValidateAudience = false,
            ValidateLifetime = false,
            ValidateIssuerSigningKey = true,

            IssuerSigningKey = new
```

```
SymmetricSecurityKey(Encoding.UTF8.GetBytes(builder.Configuration["TokenKey"]!)),
        };
    });
```

Настройка Swagger для работы с jwt

```
builder.Services.AddSwaggerGen(c =>
{

    c.SwaggerDoc("v1", new OpenApiInfo { Title = "Приложение", Version = "v2024"
});
    c.AddSecurityDefinition("Bearer", new OpenApiSecurityScheme
    {
        Description = "Authorization using jwt token. Example: \"Bearer
{token}\"",
        Name = "Authorization",
        In = ParameterLocation.Header,
        Type = SecuritySchemeType.ApiKey
    });
    c.AddSecurityRequirement(new OpenApiSecurityRequirement
    {
        {
            new OpenApiSecurityScheme
            {
                Reference = new OpenApiReference
                {
                    Type = ReferenceType.SecurityScheme,
                    Id = "Bearer"
                }
            },
            new string[] { }
        }
    });
});
```

- зарегистрируйте сервис авторизации `builder.Services.AddAuthorization();`, а затем подключите его в конвеейр сразу после процесса аутентификации

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

- для удобного тестирования jwt создайте отдельный `TokenController`

```
[ApiController]
[Route("api/[controller]")]
public class TokenController : ControllerBase
{
```

```
    private readonly IConfiguration _config;
    public TokenController(IConfiguration config)
    {
        _config = config;
    }

    [HttpGet]
    public IActionResult GenerateToken()
    {
        var claims = new List<Claim> { new Claim(ClaimTypes.Name, "user") };
        var key = new
SymmetricSecurityKey(Encoding.UTF8.GetBytes(_config["TokenKey"]!));

        // создаем JWT-токен
        var jwt = new JwtSecurityToken(
            // issuer: _config["Jwt:Issuer"],
            // audience: _config["Jwt:Audience"],
            claims: claims,
            expires: DateTime.UtcNow.Add(TimeSpan.FromDays(365)),
            signingCredentials: new SigningCredentials(key,
SecurityAlgorithms.HmacSha256));

        return Ok(new JwtSecurityTokenHandler().WriteToken(jwt));
    }
}
```

- чтобы защитить конечную точку надо поставить атрибут [Authorize]. Теперь, чтобы получить доступ по конечной точке надо передать в запросе заголовка `Authorization` валидный jwt-токен формата: `Bearer {token}`. Проверить валидность токена можно на сайте - jwt.io. Тестирование удобно делать в Postman.

```
    [Authorize]
    [HttpGet]
    public ActionResult GetUser(){
        return Ok(_repo.GetUsers());
    }
```

Задание: протестируйте конечную точку создания пользователя в jwt-аутентификацией в Postman и Swagger

Extensions. Методы расширения

- создайте папку `Extensions`, в которой создайте статический класс `JwtServices`

```
public static class JwtServices
{
    public static IServiceCollection AddJwtServices(this IServiceCollection
```

```
services, IConfiguration configuration){
        services.AddSwaggerGen(c =>
                        {
                            c.SwaggerDoc("v1", new OpenApiInfo { Title =
"Приложение", Version = "v2024" });
                            c.AddSecurityDefinition("Bearer", new
OpenApiSecurityScheme
                            {
                                Description = "Authorization using jwt token.
Example: \"Bearer {token}\"",
                                Name = "Authorization",
                                In = ParameterLocation.Header,
                                Type = SecuritySchemeType.ApiKey
                            });
                            c.AddSecurityRequirement(new
OpenApiSecurityRequirement
                            {
                                {
                                    new OpenApiSecurityScheme
                                    {
                                        Reference = new OpenApiReference
                                        {
                                            Type = ReferenceType.SecurityScheme,
                                            Id = "Bearer"
                                        }
                                    },
                                    new string[] { }
                                }
                            });
                        });
        services.AddAuthentication(JwtBearerDefaults.AuthenticationScheme)
                        .AddJwtBearer(options =>
                        {
                            options.SaveToken = true;
                            options.RequireHttpsMetadata = false;
                            options.TokenValidationParameters = new
TokenValidationParameters
                            {
                                ValidateIssuer = false,
                                ValidateAudience = false,
                                ValidateLifetime = false,
                                ValidateIssuerSigningKey = true,

                                IssuerSigningKey = new
SymmetricSecurityKey(Encoding.UTF8.GetBytes(configuration["TokenKey"]!)),
                            };
                        });
        services.AddAuthorization();

        return services;
    }
}
```

Теперь мы вынесем эту логику в отдельный файл и можем подключить это все одним сервисом.

```
builder.Services.AddJwtServices(builder.Configuration);
```

Примечание: настройте уровень логирования для отслеживания процесса работы с jwt в файле `appsettings.json`

```
"Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft.AspNetCore": "Warning",
      "Microsoft.AspNetCore.Authentication": "Debug"
    }
  }
```

Задание 1: протестируйте jwt-аутентификацию с помошью http-request.