Лабораторная работа №12
Изучение методов авторизации и аутентификации в веб-приложениях

1 Цель работы

1.1 Познакомиться с методами авторизации и аутентификации в веб-приложениях.

2 Литература

2.1 Зверева В. П., Сопровождение и обслуживание программного обеспечения компьютерных систем : учебник для студ. учреждений сред. проф. Образования / В. П. Зверева, А. В. Назаров. – М. : Издательский центр «Академия», 2018. – 256 с.

3 Подготовка к работе

3.1 Повторить теоретический материал (см. п.2).

3.2 Изучить описание лабораторной работы.

4 Основное оборудование

4.1 Персональный компьютер.

5 Задание

5.1 Разработать веб-приложение (web-api), использующее авторизацию пользователя при помощи JWT с распределением прав на две роли (роли определите самостоятельно, для каждой роли должен быть доступен конкретный запрос, который не доступен для других ролей)

5.1.1 Установить пакет Microsoft.AspNetCore.Authentication.JwtBearer

5.1.2 Добавить в файл Program настройки для авторизации

```
builder.Services.AddAuthorization();
builder.Services.AddAuthentication(JwtBearerDefaults.Authenticat
ionScheme)
    .AddJwtBearer(options =>
    {
        var                          key                          =
Encoding.ASCII.GetBytes("super_secret_jwt_key_1234567890_");
        options.TokenValidationParameters          =          new
TokenValidationParameters
        {
            ValidateIssuer = true,
            ValidIssuer = "Издатель токена",
            ValidateAudience = true,
            ValidAudience = "Получатель токена",
            ValidateIssuerSigningKey = true,
            IssuerSigningKey = new SymmetricSecurityKey(key),
        };
    });
```

5.1.3 Добавить Endpoint для авторизации, в котором проверять

корректность указанных пользователем учетных данных, и в случае успеха генерировать новый JWT-токен, который также должен хранить данные о пользователе.

```
var tokenHandler = new JsonWebTokenHandler();
var key = Encoding.ASCII.GetBytes(jwtSettings.Key);
var tokenDescriptor = new SecurityTokenDescriptor
{
    Subject = new ClaimsIdentity(
    [
        new Claim(ClaimTypes.NameIdentifier, username),
        new Claim(ClaimTypes.Role, userRole),
    ]),
    Expires                                                    =
DateTime.UtcNow.AddDays(jwtSettings.AccessTokenExpirationMinutes
),
    Issuer = "Издатель токена",
    Audience = "Получатель токена",
    SigningCredentials     =     new     SigningCredentials(new
SymmetricSecurityKey(key),
SecurityAlgorithms.HmacSha256Signature)
};
var token = tokenHandler.CreateToken(tokenDescriptor);
```

5.1.4 Создать Endpoint-ы для авторизованных запросов, укажите для них атрибут [Authorize(Roles ="ИмяРоли")] для разграничения прав доступа между запросами.

5.1.5 В ответе на запрос выводить имя и роль авторизованного пользователя. Для этого необходимо в методе запроса получить HttpContext и обратиться к свойствам context.User.Identity.Name и context.User.FindFirst(ClaimTypes.Role).Value

5.2 Разработать веб-приложение (web-api), использующее авторизацию пользователя при помощи стороннего сервиса Oauth (GitHub)

5.2.1 Установить пакет Microsoft.AspNetCore.Authentication.OAuth

5.2.2 Добавить в сервисы приложения аутентификацию, указав опции для схемы аутентификации

```
options.DefaultScheme                                          =
CookieAuthenticationDefaults.AuthenticationScheme;
options.DefaultChallengeScheme = "GitHub";
```

5.2.3 Зарегистрируйте свое приложение в GitHub (см. п.9.2)

5.2.4 Добавить в сервисы приложения поддержку Cookie (AddCookie())

5.2.5 Добавить в приложение поддержку OAuth

```
.AddOAuth("GitHub", options =>
{
    //Сюда укажите данные из GitHub
    options.ClientId = "your_client_id";
    options.ClientSecret = "your_client_secret";
    //Должен совпадать с именем эндпоинта, созданного вами
    options.CallbackPath = "/signin-github";
```

```
    //тут надо поочередно указать адреса п.5.2.6
    options.AuthorizationEndpoint = "url_1";
    options.TokenEndpoint = "url_2";
    options.UserInformationEndpoint = "url_3";
    options.SaveTokens = true;

    //Здесь надо настроить Claim Actions п.5.2.7
    options.ClaimActions.MapJsonKey(ClaimTypes.Тип, "ключ_json");

    options.Events = new OAuthEvents
    {
        OnCreatingTicket = async context =>
        {
            var request = new HttpRequestMessage(HttpMethod.Get,
context.Options.UserInformationEndpoint);
            request.Headers.Authorization          =          new
System.Net.Http.Headers.AuthenticationHeaderValue("Bearer",
context.AccessToken);
            request.Headers.UserAgent.ParseAdd("ИмяПриложения");

            var            response            =            await
context.Backchannel.SendAsync(request);
            response.EnsureSuccessStatusCode();

            using    var    user    =    JsonDocument.Parse(await
response.Content.ReadAsStringAsync());
            context.RunClaimActions(user.RootElement);
        }
    };
});
```

5.2.6 В коде указать Endpoint-ы для авторизации, получения токена и информации о пользователе в соответствии с документацией https://docs.github.com/ru/apps/oauth-apps/building-oauth-apps/authorizing-oauth-apps#1-request-a-users-github-identity (адреса из запросов пунктов 1, 2 и 3)

5.2.7 Указать ClaimActions для сохранения информации о пользователе, на основе примера ответа на запрос из документации https://docs.github.com/ru/rest/users/users

5.2.8 Добавить в Api Endpoint-ы для авторизации и вывода информации об авторизованном пользователе через context.User

5.2.9 Для авторизации используйте метод HttpContext

```
await            context.ChallengeAsync("GitHub",            new
AuthenticationProperties   {   RedirectUri   =   "/адрес_запроса_с
информацией" });
```

5.2.10 Добавьте эндпоинт для выхода из аккаунта, для этого используйте метод HttpContext

```
await context.SignOutAsync();
```

6 Порядок выполнения работы

6.1 Повторить теоретический материал п. 3.1;

6.2 Выполнить задания 5.1-5.2
6.3 Ответить на контрольные вопросы п. 8;
6.4 Заполнить отчет п. 7.

7 Содержание отчета

7.1 Титульный лист;
7.2 Цель работы;
7.3 Ответы на контрольные вопросы п. 6.3;
7.4 Вывод по проделанной работе.

8 Контрольные вопросы

8.1 Что такое JWT?
8.2 Что такое Oauth?

9 Приложение

9.1 Для использования JWT в Swagger необходимо изменить код подключения Swagger:

```
builder.Services.AddSwaggerGen(c =>
{
    c.SwaggerDoc("v1", new OpenApiInfo
    {
        Title = "Мое АПИ",
        Version = "v1"
    });

    c.AddSecurityDefinition("Bearer", new OpenApiSecurityScheme
    {
        Description = "JWT Authorization header using the Bearer
scheme. Example: \"Authorization: Bearer {token}\"",
        Name = "Authorization",
        In = ParameterLocation.Header,
        Type = SecuritySchemeType.Http,
        Scheme = "bearer"
    });
    c.AddSecurityRequirement(new OpenApiSecurityRequirement {
        {
            new OpenApiSecurityScheme {
                Reference = new OpenApiReference {
                    Type = ReferenceType.SecurityScheme,
                    Id = "Bearer"
                }
            },
            new string[] {}
        }
    });
});
```

9.2 Регистрация приложения в GitHub

1. Перейдите в настройки разработчика GitHub: GitHub Developer Settings. https://github.com/settings/developers

2. Создайте новое приложение (New OAuth App):

• Application name: Укажите имя вашего приложения.

• Homepage URL: Укажите URL вашего приложения (например, http://localhost:5000 для локальной разработки).

• Authorization callback URL: Укажите путь для перенаправления авторизации (например, http://localhost:5000/signin-github).

3. После регистрации сохраните:

• Client ID.

• Client Secret (необходимо будет сгенерировать).