**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**



**МОСКОВСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

***Факультет Информационных технологий***

***Кафедра Информатики и информационных технологий***

**направление подготовки**

**09.03.02 «Информационные системы и технологии»**

**ЛАБОРАТОРНАЯ РАБОТА №** 15

**Дисциплина: «Backend»**

**Тема:** *Реализация механизма перенаправления запросов через middleware в веб-приложении на платформе ASP.NET Core*

**Выполнил: студент группы: 231-339**

Карапетян Нвер Каренович\_\_\_\_\_\_\_\_\_\_\_

(Фамилия И.О.)



**Дата, подпись:** 14.05.25 \_\_\_\_\_\_\_\_\_\_\_\_

(Дата)(Подпись)

**Проверил:** \_\_*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_*

(Фамилия И.О., степень, звание) (Оценка)

**Дата, подпись** \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ *\_\_\_\_\_\_\_\_\_\_\_\_\_*

(Дата)(Подпись)

**Москва**

**2025**

Цель:

Научиться реализовывать механизмы перенаправления запросов через промежуточное программное обеспечение (middleware) для контроля доступа к различным адресам в веб-приложении на платформе ASP.NET Core.

Задачи:

* Создать роли и определить права доступа для пользователей в приложении.
* Реализовать middleware для перенаправления запросов, основываясь на авторизации и аутентификации пользователей с использованием [ASP.NET](https://asp.net/) Core Identity или других подходящих механизмов.
* Настроить middleware для контроля доступа, ограничивая доступ к определенным адресам или контроллерам в зависимости от ролей или полномочий пользователей.
* Провести тестирование механизма перенаправления для проверки доступа к различным адресам для пользователей с разными ролями.

Ход работы

В современных веб‑приложениях часто требуется защищать отдельные части API и предоставлять доступ к ним только определенным категориям пользователей. В данной работе рассматривался сценарий, в котором существует единая точка входа /api/admin, к которой должны получать доступ только пользователи с ролью «**Admin»**. Все остальные обращения к этому адресу автоматически перенаправляются на страницу отказа в доступе /api/forbidden. Важное требование: проверка прав должна происходить централизованно, до выполнения бизнес‑логики в контроллерах, то есть на уровне middleware ASP.NET Core.

Настройка службы Identity и контекста базы данных

Для обеспечения хранения пользователей и ролей применен IdentityDbContext<AppUser>. В методе OnModelCreating были зафиксированы две базовые роли, которые автоматически вносятся в таблицу AspNetRoles при выполнении миграций:

****Листинг 1.**** ApplicationDbContext.OnModelCreating.

protected override void OnModelCreating(ModelBuilder builder)

{

    base.OnModelCreating(builder);

    List<IdentityRole> roles = new List<IdentityRole>

    {

        new IdentityRole

        {

            Id = "Admin\_ID",

            Name = "Admin",

            NormalizedName = "ADMIN"

        },

        new IdentityRole

        {

            Id = "User\_ID",

            Name = "User",

            NormalizedName = "USER"

        }

    };

    builder.Entity<IdentityRole>().HasData(roles);

}

После выполнения создания миграции («Add-Migration Initial») и обновления базы данных («Update-Migration») в базе автоматически создаются все необходимые таблицы для Identity, а также две роли.

Настройка JWT‑аутентификации и генерация токенов

Далее в Program.cs была сконфигурирована аутентификация на основе JWT. Параметры валидности токена (Issuer, Audience, SigningKey) берутся из конфигурации, а RoleClaimType установлен в ClaimTypes.Role, что позволит стандартным методам User.IsInRole() корректно считывать роли из токена.

Листинг 2. Конфигурация JWT в Program.cs.

builder.Services.AddAuthentication(options =>

{

    options.DefaultAuthenticateScheme =

    options.DefaultChallengeScheme =

    options.DefaultForbidScheme =

    options.DefaultScheme =

    options.DefaultSignInScheme =

    options.DefaultSignOutScheme = JwtBearerDefaults.AuthenticationScheme;

}).AddJwtBearer(options =>

{

    options.TokenValidationParameters = new TokenValidationParameters

    {

        ValidateIssuer = true,

        ValidIssuer = builder.Configuration["Jwt:Issuer"],

        ValidateAudience = true,

        ValidAudience = builder.Configuration["Jwt:Audience"],

        ValidateIssuerSigningKey = true,

        RoleClaimType = ClaimTypes.Role,

        IssuerSigningKey = new SymmetricSecurityKey(System.Text.Encoding.UTF8.GetBytes(builder.Configuration["Jwt:SigningKey"]))

    };

});

В сервис JwtTokenService была интегрирована логику добавления ролей пользователя в список claim’ов при генерации токена:

Листинг 3. Метод CreateToken в JwtTokenService.

public AuthResponse CreateToken(AppUser user)

{

    var claims = new List<Claim>

    {

        new Claim(JwtRegisteredClaimNames.Email, user.Email),

        new Claim(JwtRegisteredClaimNames.GivenName, user.UserName)

    };

    var roles = \_userManager.GetRolesAsync(user).GetAwaiter().GetResult();

    foreach (var role in roles)

        claims.Add(new Claim(ClaimTypes.Role, role));

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

    var tokenDescriptor = new SecurityTokenDescriptor

    {

        Subject = new ClaimsIdentity(claims),

        Expires = DateTime.Now.AddDays(1),

        SigningCredentials = creds,

        Issuer = \_config["Jwt:Issuer"],

        Audience = \_config["Jwt:Audience"]

    };

    var tokenHandler = new JwtSecurityTokenHandler();

    var token = tokenHandler.CreateToken(tokenDescriptor);

    return new AuthResponse

    {

        Email = user.Email,

        Token = tokenHandler.WriteToken(token),

        Expires = token.ValidTo

    };

}

Реализация контроллера аутентификации

Контроллер AuthController обслуживает регистрацию и вход пользователей. При успешном входе метод Login возвращает клиенту сформированный JWT.

****Листинг 4.**** Метод Login в AuthController.

[HttpPost("login")]

public async Task<IActionResult> Login(LoginDto dto)

{

    if (!ModelState.IsValid)

        return BadRequest(ModelState);

    var user = await \_userManager.Users.FirstOrDefaultAsync(u => u.Email == dto.Email);

    if (user is null)

        return Unauthorized("Invalid username!");

    var result = await \_signInManager.CheckPasswordSignInAsync(user, dto.Password, false);

    if (!result.Succeeded)

        return Unauthorized("Username is not found and/or password is incorrect!");

    return Ok(\_tokenService.CreateToken(user));

}

Примерно аналогично метод Register, но с обработкой исключений, создает нового пользователя, присваивает ему роль «**User»** и возвращает JWT:

****Листинг 4.**** Метод Login в AuthController.

****Листинг 5.**** Метод Register в AuthController.

[HttpPost("register")]

public async Task<IActionResult> Register([FromBody] RegisterDto dto)

{

    try

    {

        if (!ModelState.IsValid)

            return BadRequest(ModelState);

        var user = new AppUser

        {

            UserName = dto.Username,

            Email = dto.Email,

        };

        var createdUser = await \_userManager.CreateAsync(user, dto.Password);

        if (createdUser.Succeeded)

        {

            var roleResult = await \_userManager.AddToRoleAsync(user, "User");

            if (roleResult.Succeeded)

                return Ok(\_tokenService.CreateToken(user));

            else

                return StatusCode(500, roleResult.Errors);

        }

        else

            return StatusCode(500, createdUser.Errors);

    }

    catch (Exception e)

    {

        return StatusCode(500, e);

    }

}

Разработка Middleware для перенаправления запросов

Ключевой компонент — класс RoleRedirectMiddleware, который перехватывает каждый запрос **после** аутентификации, но **до** вызова контроллеров. Если путь начинается с /api/admin, middleware проверяет, залогинен ли пользователь, и обладает ли он ролью «**Admin»**. При несоответствии выполняется Redirect на /api/forbidden.

****Листинг 6.**** Класс RoleRedirectMiddleware.

public class RoleRedirectMiddleware

{

    private readonly RequestDelegate \_next;

    public RoleRedirectMiddleware(RequestDelegate next)

    {

        \_next = next;

    }

    public async Task InvokeAsync(HttpContext context)

    {

        var path = context.Request.Path.Value?.ToLower() ?? "";

        if (path!.StartsWith("/api/admin"))

        {

            if (!context.User.Identity!.IsAuthenticated)

            {

                context.Response.Redirect("/api/forbidden");

                return;

            }

            if (!context.User.IsInRole("Admin"))

            {

                context.Response.Redirect("/api/forbidden");

                return;

            }

        }

        await \_next(context);

    }

}

В Program.cs это middleware подключено сразу после UseAuthorization() и до MapControllers():

****Листинг 7. Подключение middleware****.

app.UseAuthentication();

app.UseAuthorization();

app.UseMiddleware<RoleRedirectMiddleware>();

app.MapControllers();

Контроллеры защищенных ресурсов и страницы отказа

После middleware запрос либо поступает в AdminController, либо перенаправляется:

****Листинг 8. AdminController****.

[Route("api/[controller]")]

[ApiController]

public class AdminController : ControllerBase

{

    [HttpGet]

    public IActionResult Index()

    {

        return Content("Welcome to the Admin page", "text/plain");

    }

}

****Листинг 9. ForbiddenController****.

[Route("api/[controller]")]

[ApiController]

public class ForbiddenController : ControllerBase

{

    [HttpGet]

    public IActionResult Index()

    {

        return StatusCode(403, "Access Denied");

    }

}

При редиректе клиент получает код 302 с Location /api/forbidden, а при запросе к /api/forbidden — ответ 403 Forbidden.

Тестирование

Проверка выполнялась с помощью HTTP-клиента Scalar. В трёх сценариях получены следующие результаты:

1. **Без токена**

Запрос к /api/admin возвращает 302 Redirect —> /api/forbidden.

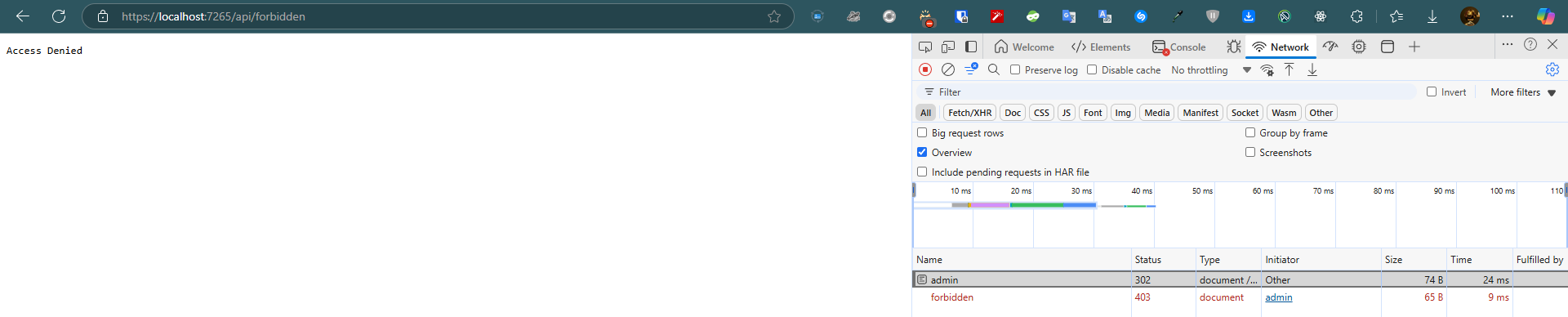


Рисунок . Запрос к /api/admin без токена.

1. **С токеном обычного пользователя**

Токен без роли «**Admin»** приводит к тому же перенаправлению.

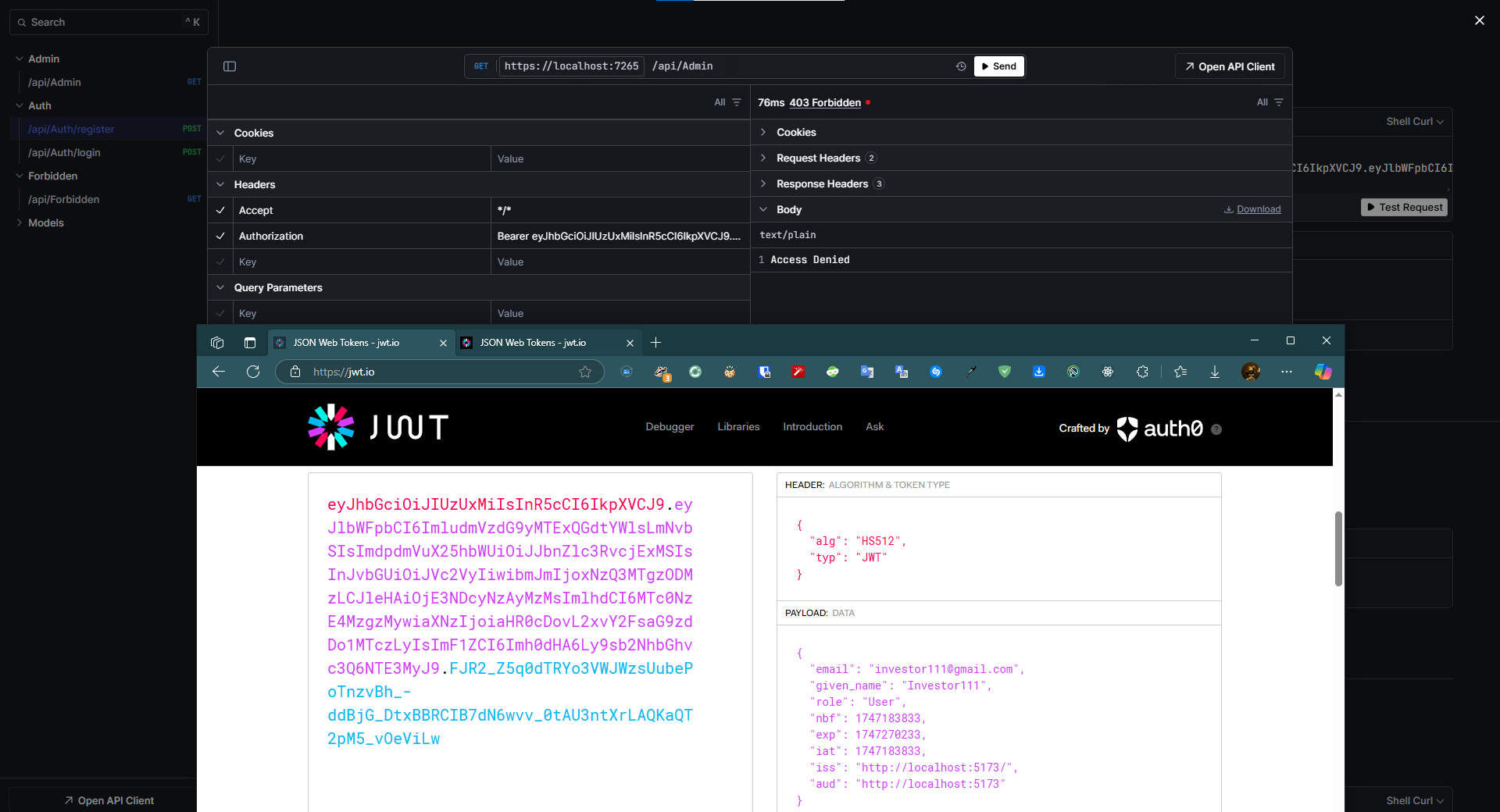


Рисунок . Запрос к /api/admin от пользователя с ролью «User».

1. **С токеном администратора**

Запрос к /api/admin возвращает 200 OK и содержимое контроллера.

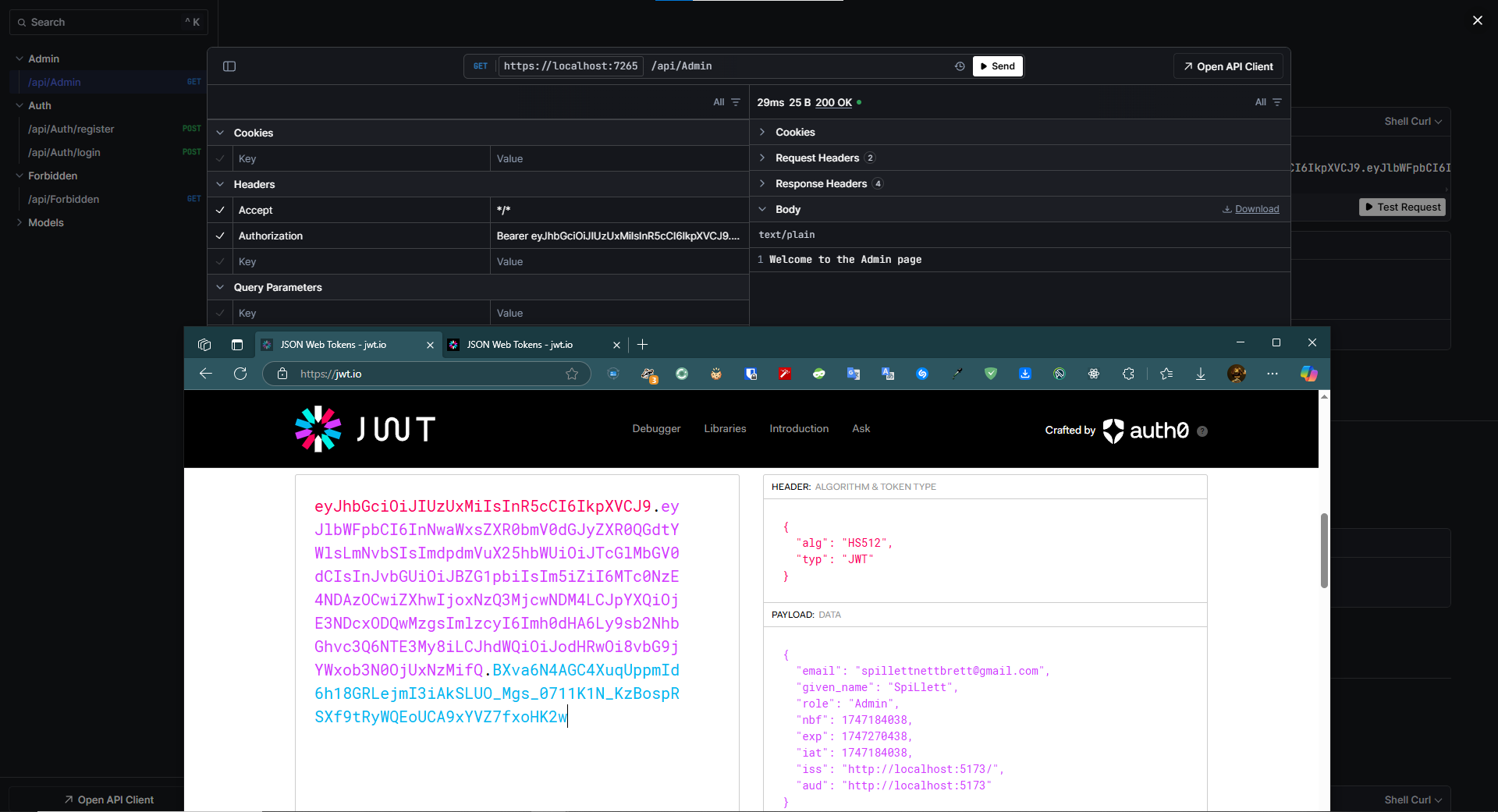


Рисунок . Запрос к /api/admin от админа.

Запрос к /api/forbidden на любом шаге возвращает 403 Forbidden с текстом «Access Denied».