Лабораторная работа №6

Аутентификация и авторизация

1. Цель работы.

Знакомство с механизмом аутентификации и авторизации в ASP.Net Core.

2. Задача работы

Научиться выполнять аутентификацию и авторизацию на удаленном сервере. Научиться ограничивать доступ к Api для незарегистрированных пользователей.

Время выполнения работы: 6 часов (3 занятия)

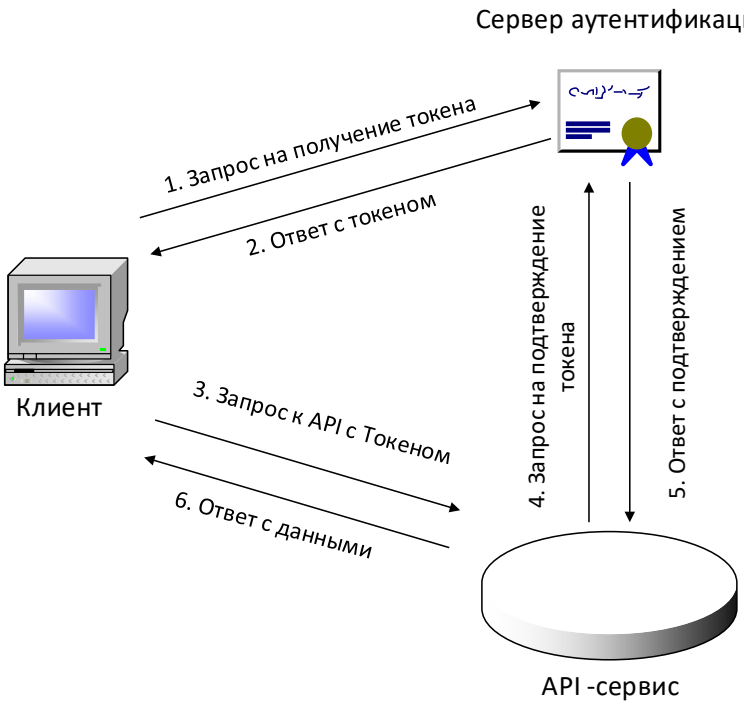
3. Предварительная информация

3.1. Выбор сервера аутентификации

Для более детального ознакомления с аутентификацией и авторизацией в ASP.NET Core в данной работе предлагается использовать отдельное приложение – сервер аутентификации. Данный сервер должен предоставить страницы входа в систему, регистрации в системе, предоставление токена аутентификации и возможность проверки токена.

Поскольку доступ к готовым серверам аутентификации, таким как Azure Active Directory, ограничен в связи с санкциями, предлагается создать свое приложение – сервер аутентификации. В качестве такого сервера воспользуемся готовым решением – Keycloak (https://www.keycloak.org/)

Схема взаимодействия с сервером аутентификации при работе с API выглядит так:



3.2. Некоторые термины Keycloak

3.2.1. Realm

В контексте Keycloak Realm представляет собой домен безопасности и администрирования, в котором осуществляется управление пользователями, приложениями и ролями. Это фундаментальная концепция архитектуры Keycloak, которая позволяет изолировать и организовывать ресурсы, разрешения и конфигурации.

3.2.2. API scope

Первоначальная спецификация OAuth 2.0 имеет концепцию областей (scope), которая просто определяется как область доступа, которую запрашивает клиент при работе с API.

Пример регистрации областей:

```
public static IEnumerable<ApiScope> ApiScopes =>
     new ApiScope[]
     {
          new ApiScope("scope1"),
          new ApiScope("scope2"),
     };
```

3.2.3. Clients

В Keycloak Client относится к приложению или сервису, который взаимодействует с сервером Keycloak для целей аутентификации и авторизации. Это может быть веб-приложение, мобильное приложение, серверный API или любой другой тип приложения, которому необходимо аутентифицировать и авторизовать своих пользователей.

Клиенты создаются внутри Realm.

4. Выполнение работы.

4.1. Предварительная информация

Для использования Keycloak можно:

- установить сервер Keycloak и сервер базы данных на компьютер
- использовать готовые образы Keycloak и сервера БД для Docker.

В настоящем руководстве описаны оба способа. Используйте любой из них.

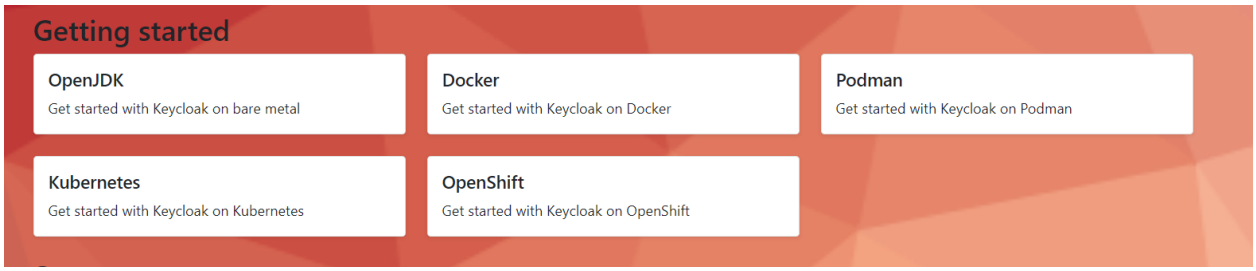
> В приведенных примерах используется PostgreSQL. Вы можете использовать любую другую базу данных

4.2. Локальная установка Keycloak и сервера БД

4.2.1. Установка Keycloak

Документация Keycloak на сайте https://www.keycloak.org/guides

Выберите способ запуска сервера Keycloak:



В приведенных примерах будет использоваться OpenJDK (https://www.keycloak.org/getting-started/getting-started-zip)

Установка OpenJDK (можно использовать любую другую Java-машину) - см. https://learn.microsoft.com/ru-ru/java/openjdk/install)

Назначьте сертификат SSL

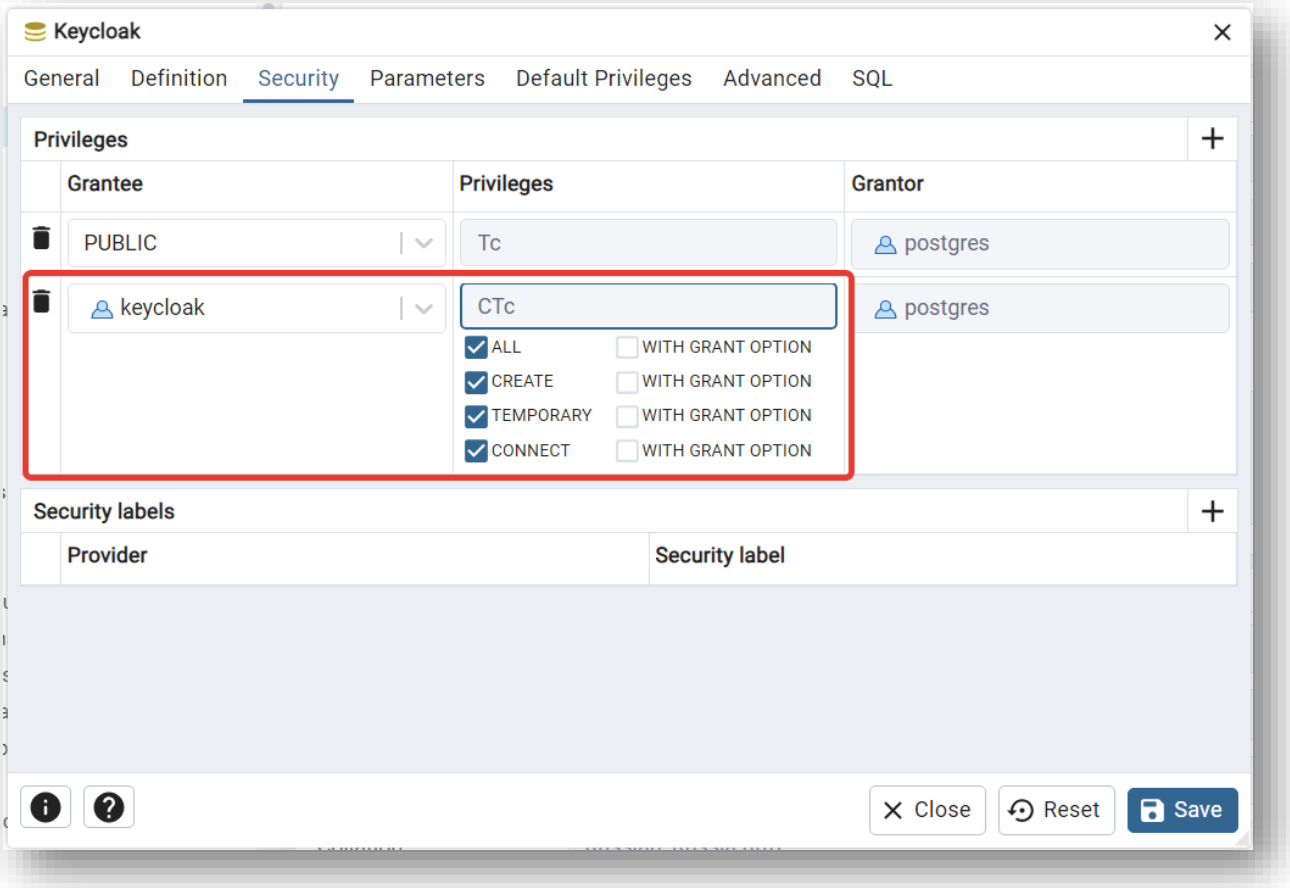
4.2.2. Подключение базы данных к Keycloak

Загрузите и установите PostgreSql (https://www.postgresql.org/download/)

Создайте базу данных (например Keycloak)

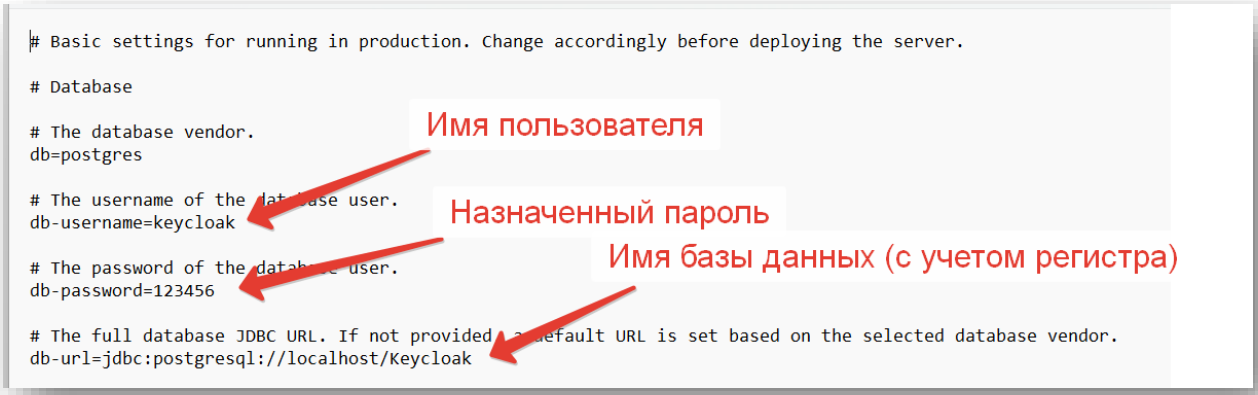
Создайте пользователя (например keycloak)

Добавьте созданного пользователя в базу данных и дайте ему все права:



Укажите использование базы данных в keycloak. Для этого откройте файл

в папке keycloak и раскомментируйте строки как показано на рисунке, с указанием ваших данных:




```
# Basic settings for running in production. Change accordingly before deploying the server.

# Database

# The database vendor.
db=postgres

# The username of the database user.
db-username=keycloak

# The password of the database user.
db-password=123456

# The full database JDBC URL. If not provided, a default URL is set based on the selected database vendor.
db-url=jdbc:postgresql://localhost/Keycloak
```

Для удобства запуска Keycloak можно создать файл xxx.bat (xxx- любое удобное для вас имя) с таким содержимым:

start bin\kc.bat start-dev

Запустите Keycloak.

4.3. Запуск Keycloak и сервера БД в Docker

Остановите работу контейнеров postgres и pgadmin командой

docker compose down.

Добавьте в файл compose.yaml сервис keycloak. Пример:

```
keycloak:
    image: quay.io/keycloak/keycloak:26.0.4
    container_name: keycloak
    command: start-dev
    environment:
      KC_DB: postgres
      KC_DB_URL_HOST: postgres
      KC_DB_URL_PORT: 5432
      KC_DB_USERNAME: ${POSTGRES_USER}
      KC_DB_PASSWORD: ${POSTGRES_PASSWORD}
      KEYCLOAK_ADMIN: ${KEYCLOAK_ADMIN}
      KEYCLOAK_ADMIN_PASSWORD: ${KEYCLOAK_ADMIN_PASSWORD}
    ports:
      - "8080:8080"
    networks:
      - postgres_network
    depends_on:
      - postgres
    restart: unless-stopped
```

Отредактируйте файл файл *.env:

POSTGRES_USER=admin

POSTGRES_PASSWORD=123456

POSTGRES_DB=keycloak

PGADMIN_DEFAULT_EMAIL=admin@example.com

PGADMIN_DEFAULT_PASSWORD=123456

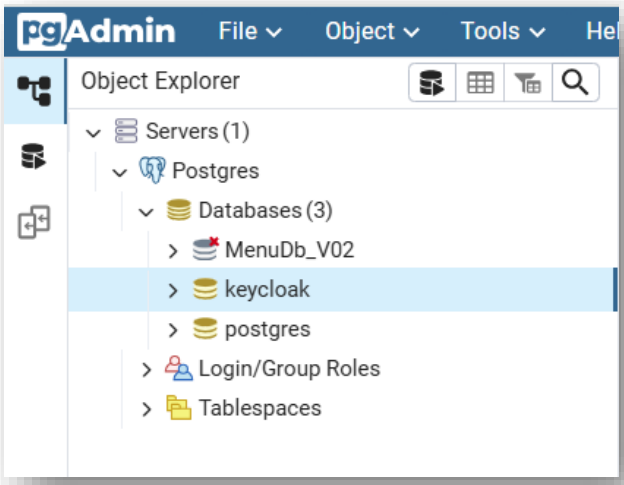
KEYCLOAK_ADMIN=admin

KEYCLOAK_ADMIN_PASSWORD=123456

Примечание. Для хранения данных keycloak должна создаться база данных с именем, указанном в переменной POSTGRES_DB. Поскольку данные хранятся на диске (см. раздел `volumes` в файле compose.yaml), а мы остановили контейнеры командой «docker compose down», то данные не уничтожаются, но и новая БД не будет создана автоматически. Для создания контейнеров «с нуля» требуется команда «**docker compose down -v**», но это уничтожит все данные. Поэтому для работы keycloak нужно создать базу данных вручную в PgAdmin4.

Запустите проект (docker compose --env-file xxx.env up -d)

Зайдите в PgAdmin. Подключитесь к серверу Postgres и **добавьте базу данных** с именем, указанном в переменной POSTGRES_DB (в примере – keycloak).



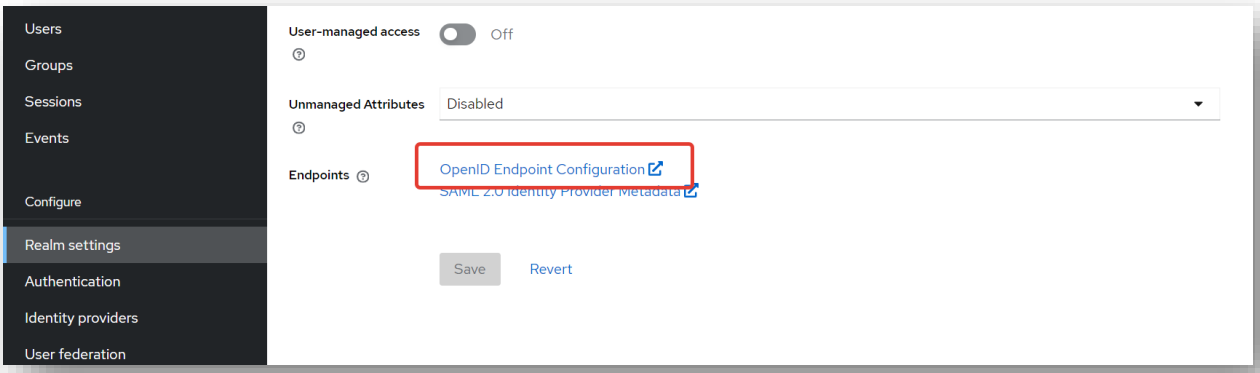
4.4. Настройка Keycloak

4.4.1. Создание Realm.

Войдите на сервер KeyCloak (localhost:8080) и создайте Realm.

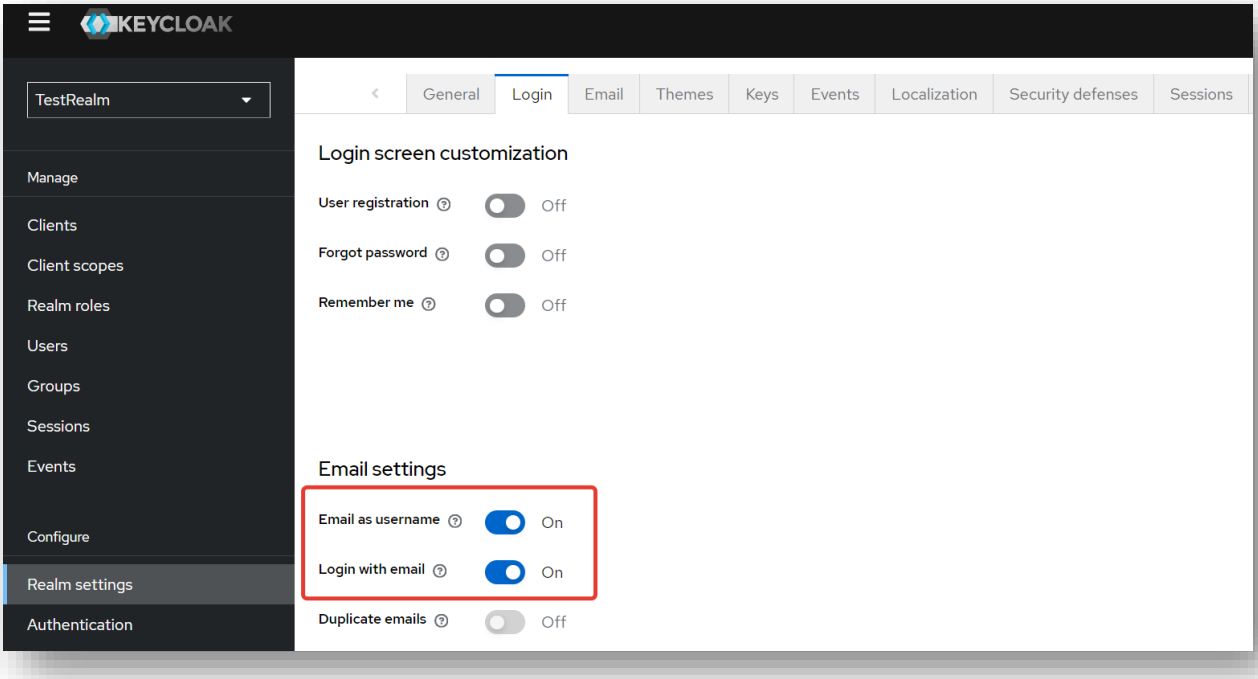
В качестве названия для Realm используйте вашу фамилию

Откройте вкладку Realm Settings. Кликните ссылку конечных точек.



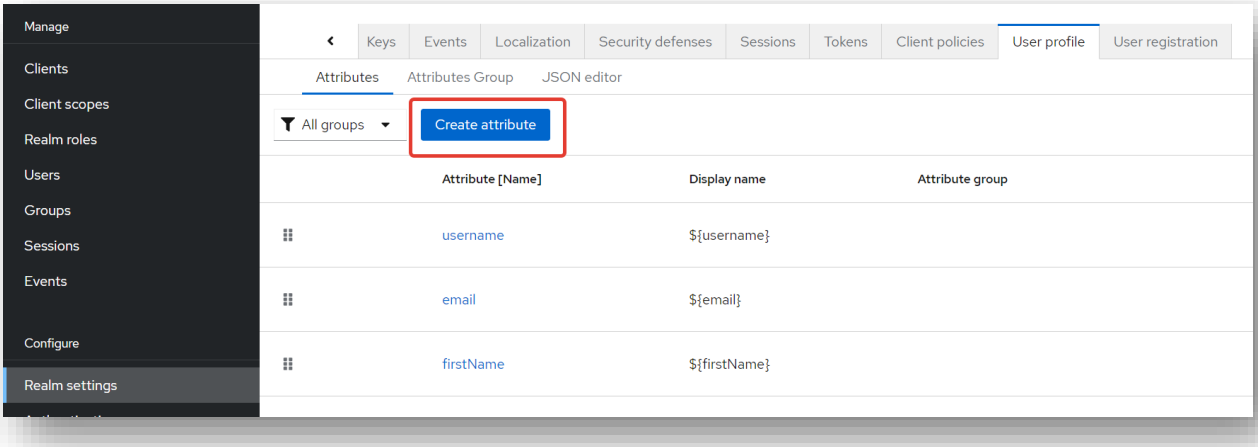
Найдите адреса конечных точек для доступа к вашему Realm.

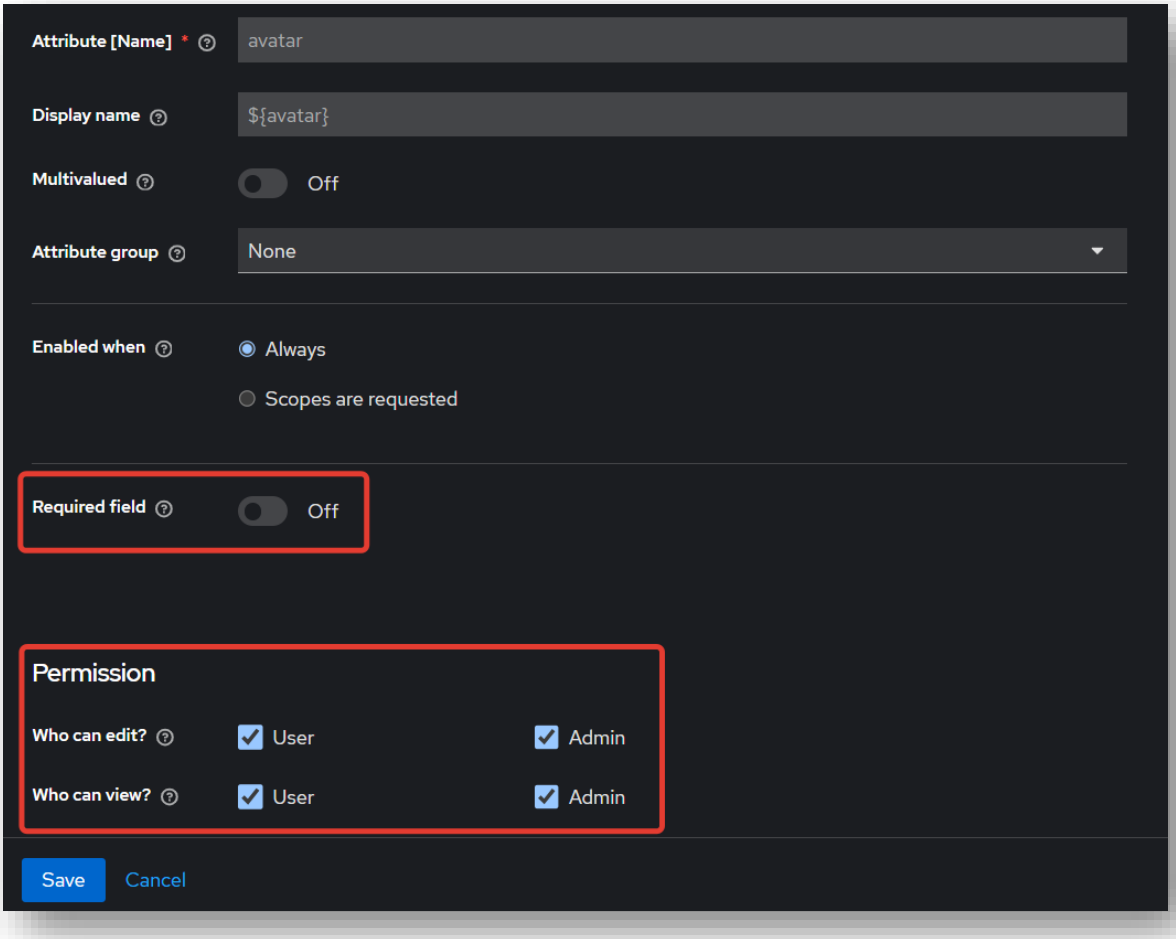
На вкладке Login установите использование email в качестве имени пользователя:



На вкладке User Profile удалите поля FirstName и LastName (мы не будем их использовать)

На вкладке User Profile добавьте для пользователя необязательный атрибут avatar для хранения Url аватара пользователя.

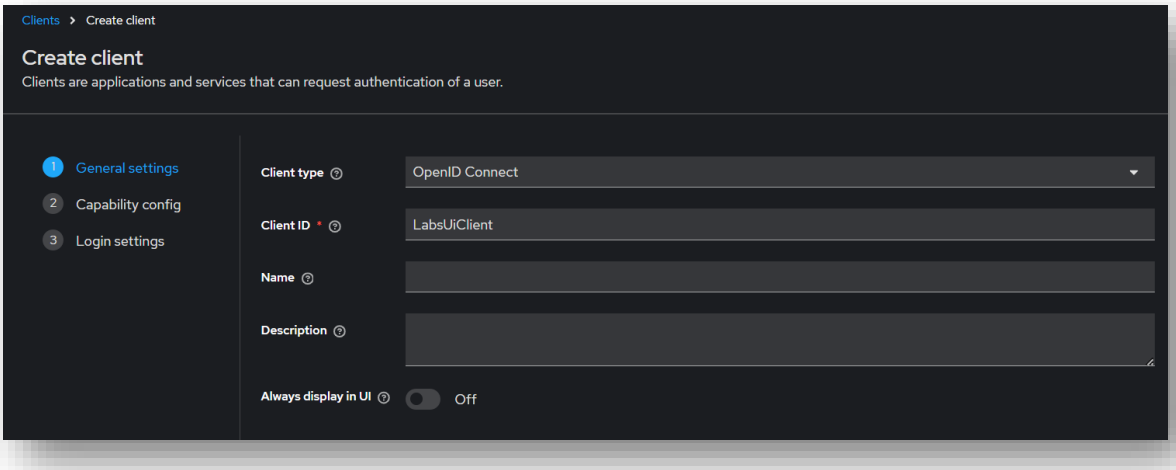




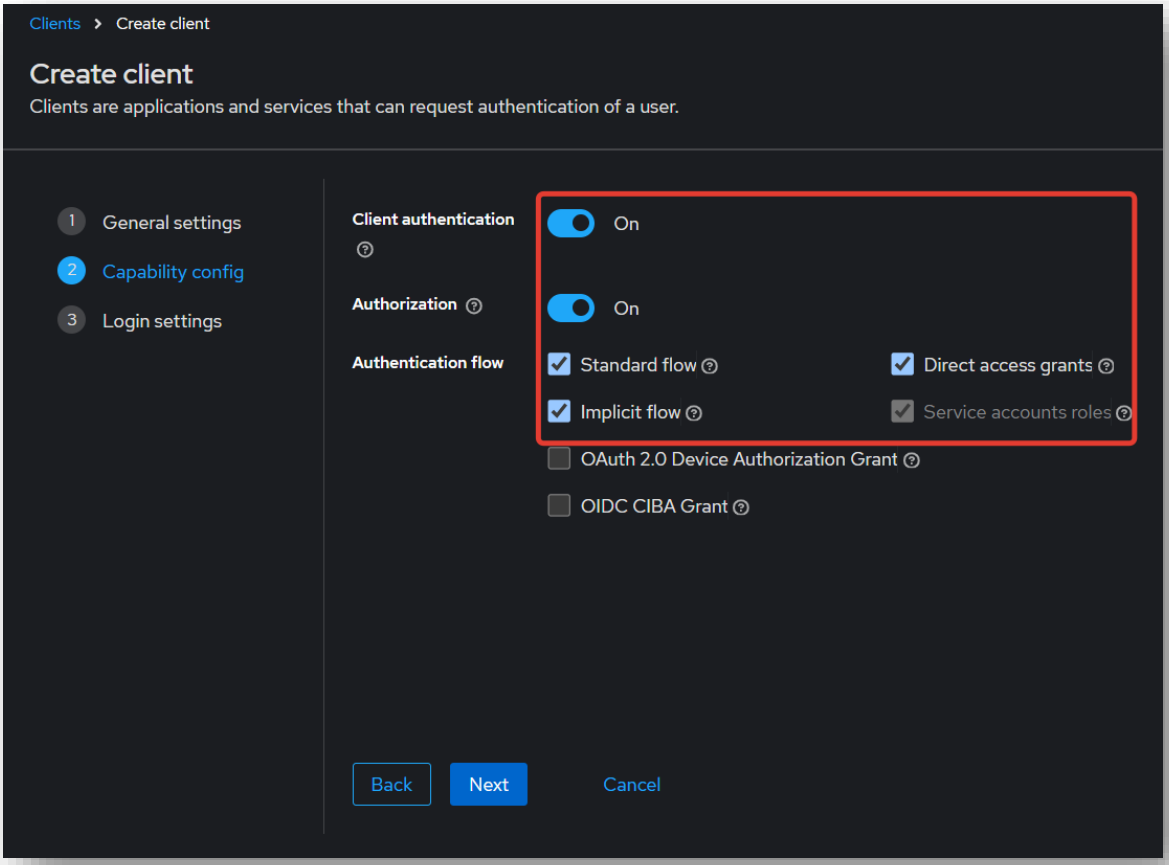
4.4.2. Создание клиента.

На вкладке Clients создайте клиента типа OpenID Connect.

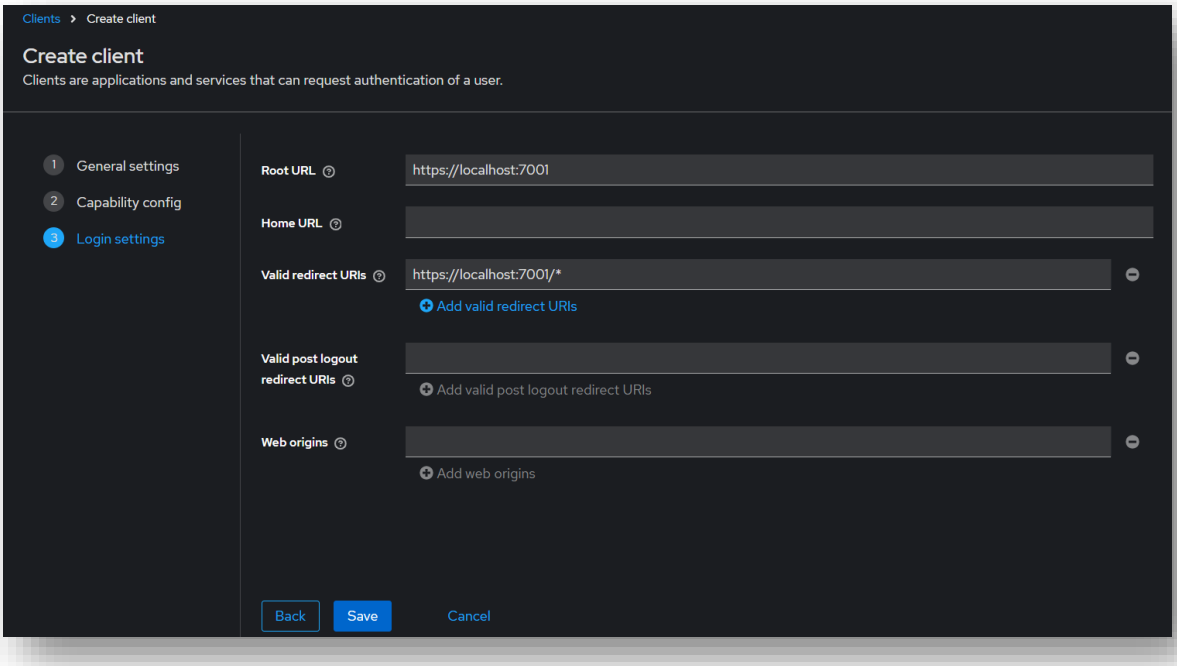
Используйте название [Ваша фамилия]UiClient



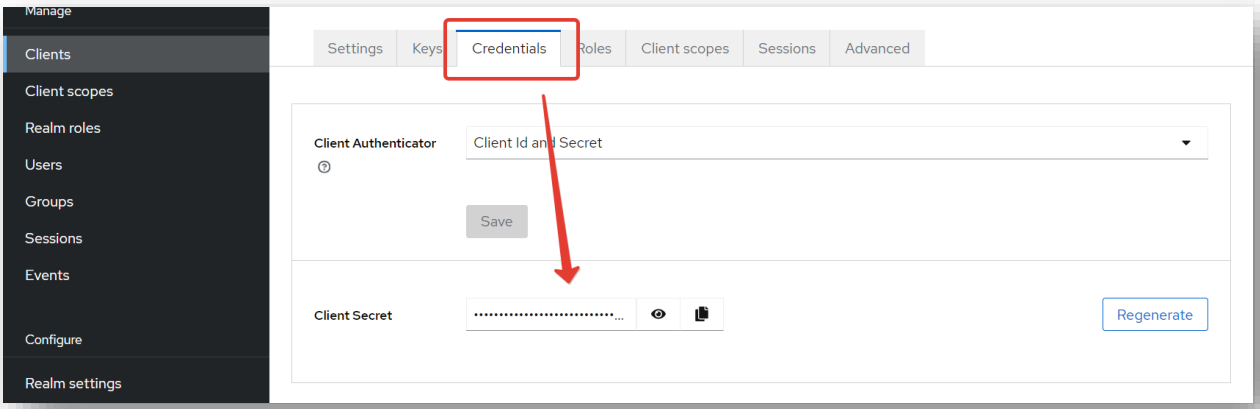
Разрешите Client Authentication, Authorization и Implicit Flow:



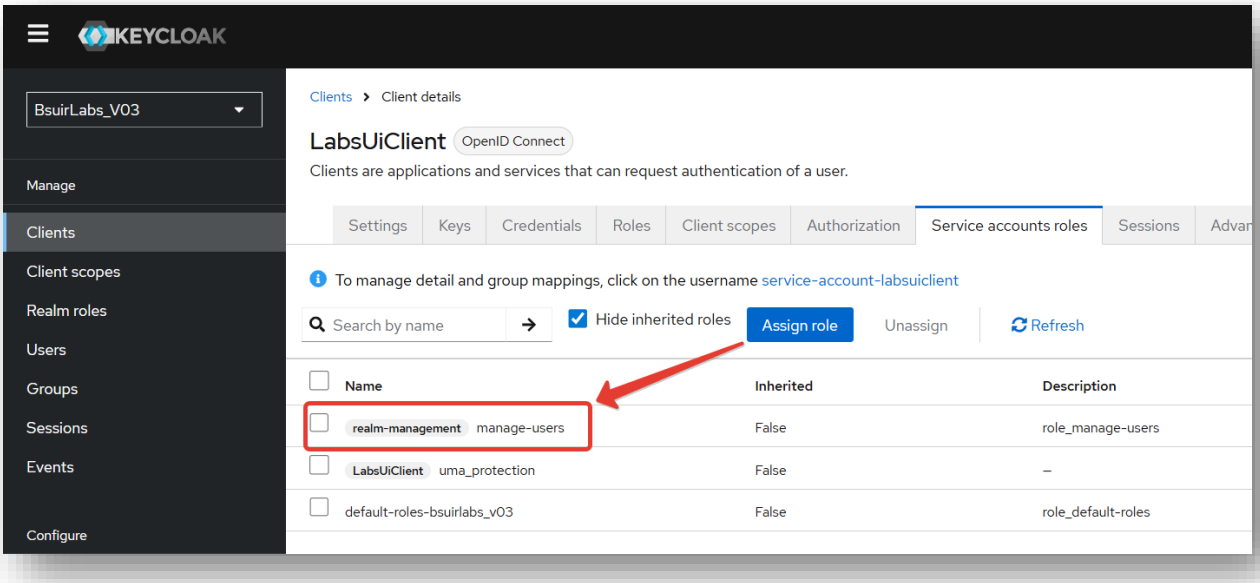
Укажите Url вашего приложения:



На вкладке Credentials найдите поле Client Secret



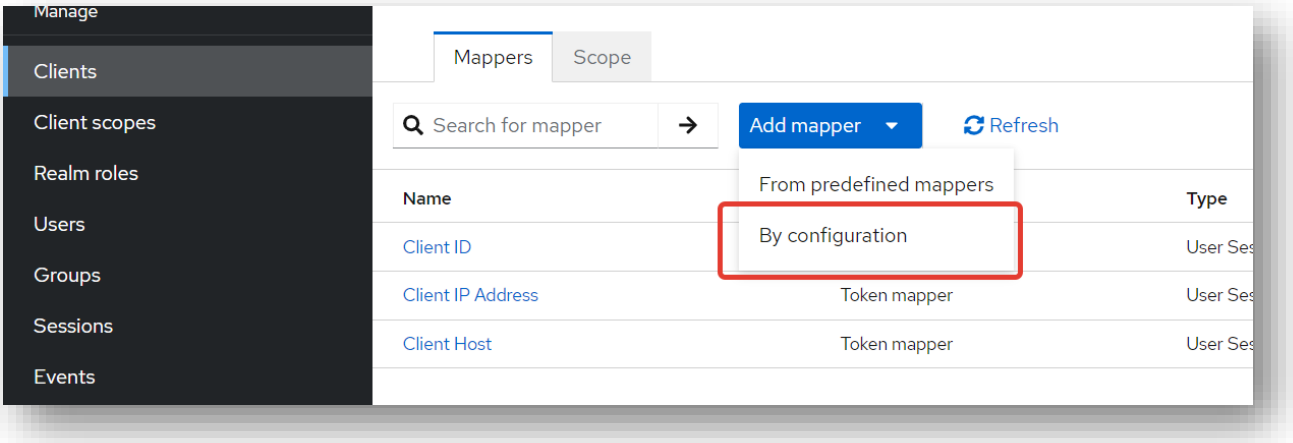
В списке клиентов выберите созданного клиента. На вкладке «Service Account roles» назначьте роль **manage-users,** чтобы приложение могло регистрировать новых пользователей



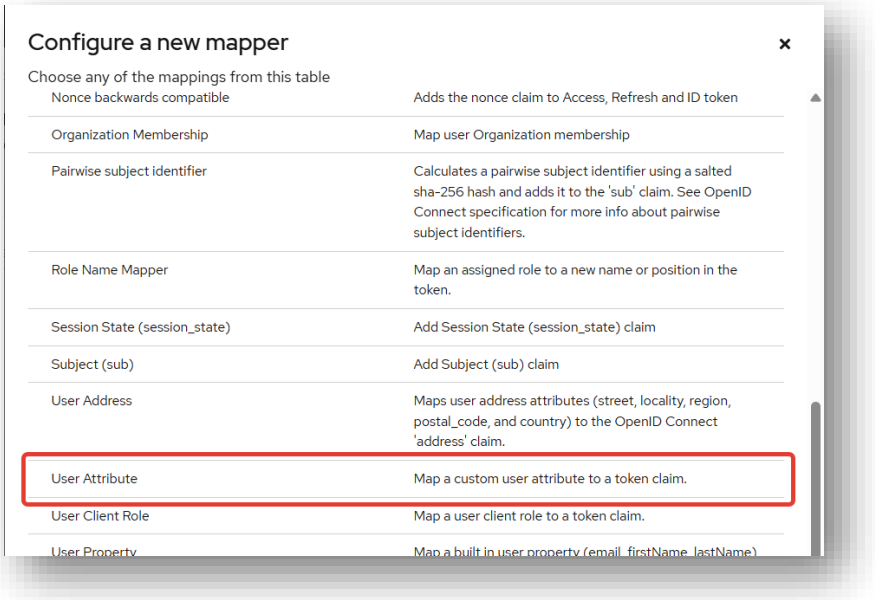
4.4.3. Передача атрибута «Avatar» в JWT токене

На вкладке ClientScopes выберите xxx-dedicated, где XXX – имя вашего клиента.

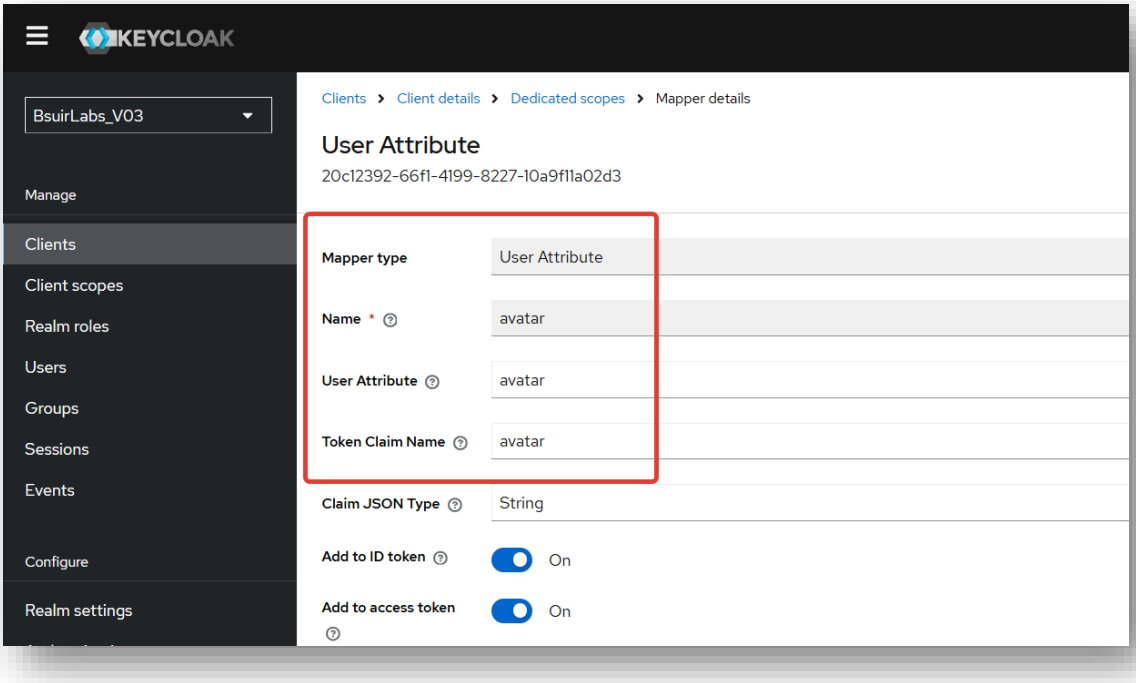
Выберите Add mapper -> By configuration



Выберите в списке User Attribute



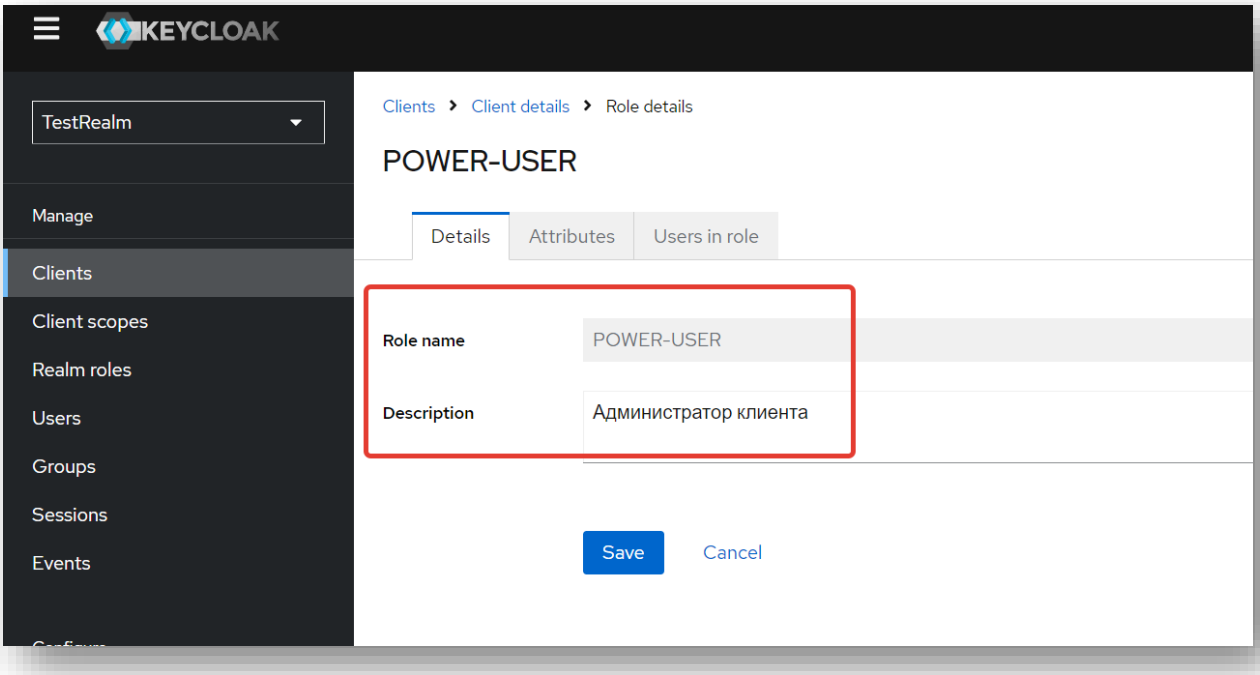
Выберите атрибут avatar



Теперь поле avatar будет передаваться вместе с JWT-токеном

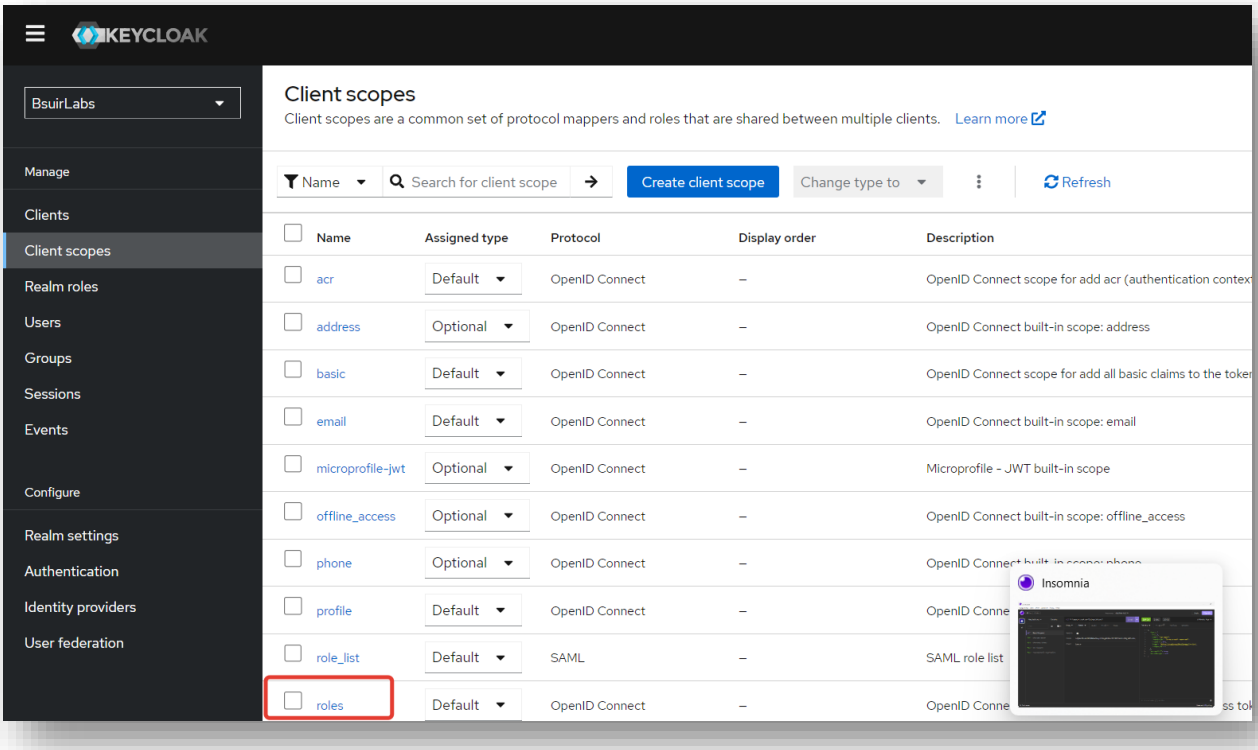
4.4.4. Создание роли администратора

На вкладке Roles добавьте роль POWER-USER для администратора вашего приложения:

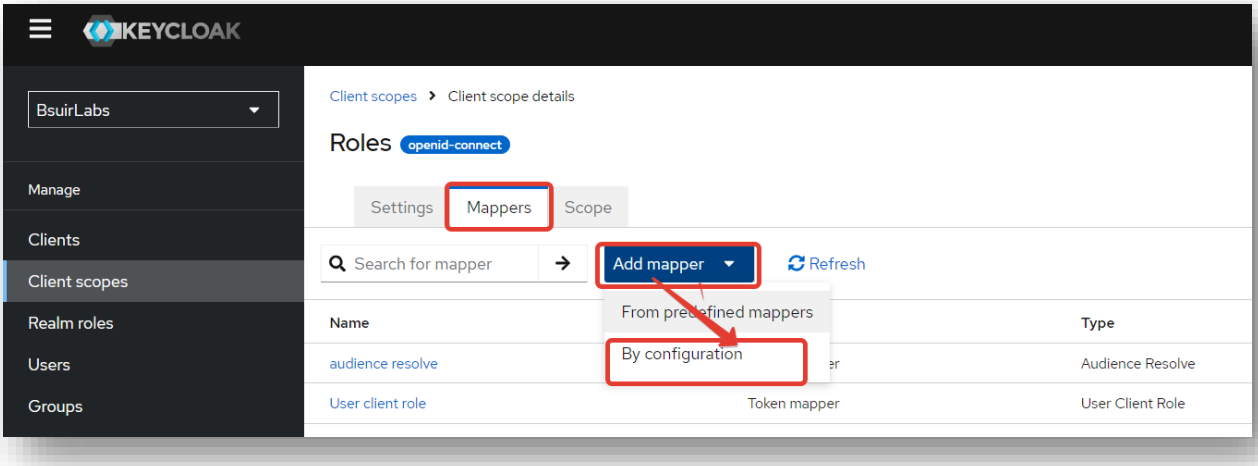


4.4.5. Передача роли в токене в виде Claim

На вкладке «Client scopes» выберите «roles»

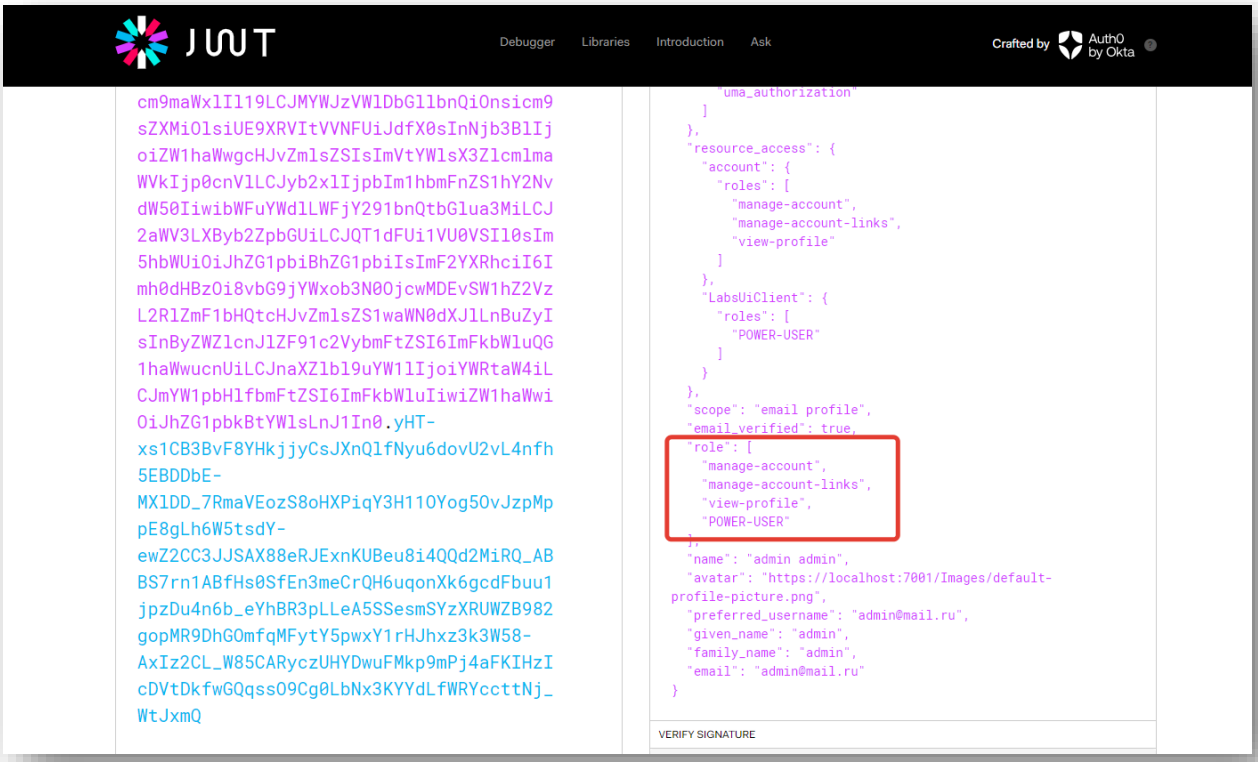


На вкладке «Mappers» добавьте Mapper «By configuration». Выберите из списка: «User client role».



- «Multivated» должно быть «On»
- «Token claim name» должно быть «role»
- «Add to access token» должно быть «On»

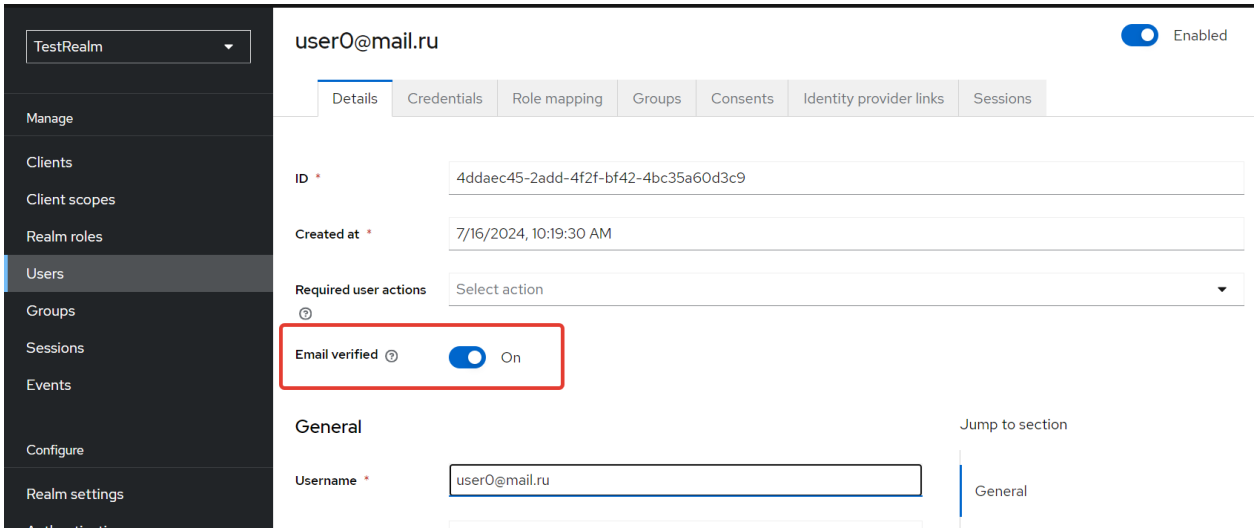
Теперь роли пользователя будут передаваться вместе с токеном в виде Claim.



4.4.6. Создание пользователей

На вкладке «Users» создайте пользователя

Укажите, что электронный адрес подтвержден



На вкладке Credentials укажите пароль пользователя. Укажите, что пароль не временный.

Создайте ***еще одного*** пользователя – администратора вашего приложения. Выберите созданного пользователя в списке пользователей. На вкладке «Role Mapping» назначьте пользователю созданную роль POWER-USER.

4.5. Проверка работы сервера Keycloak

Используйте **Postman** или **Insomnia**.

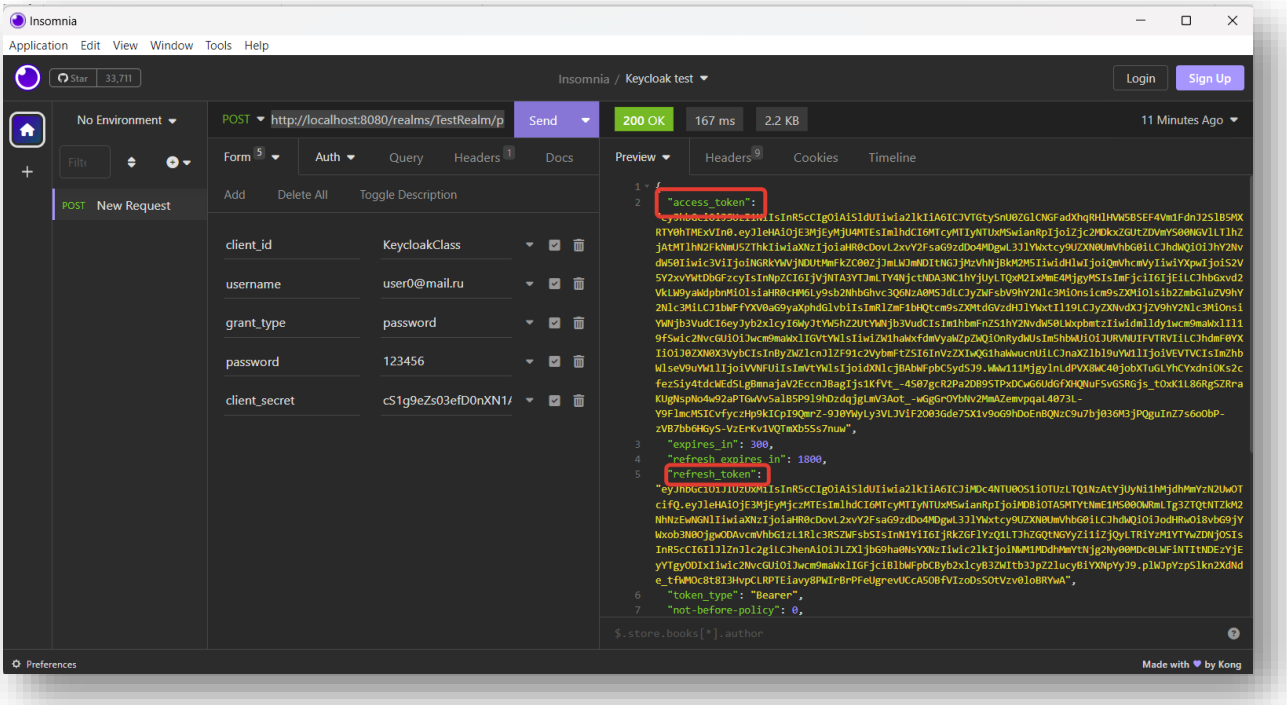
Выполните запрос по методу Post к конечной точке **`token_endpoint`**

В теле формы укажите:

- client_id – Id созданного клиента
- u
- grant_type – password
- p
- password – пароль созданного пользователя
- c
- client_secret – client secret для созданного клиента

username – электронный адрес созданного пользователя

В ответ вы должны получить JWT-токен ***пользователя***:



Измените тело формы:

- client_id – Id созданного клиента
- grant_type – client_credentials
- client_secret – client secret для созданного клиента

Отправьте запрос. В ответ должен быть получен JWT-токен для ***клиента***

4.6. Настройка проекта XXX.API

Добавьте в проект XXX.Api пакет NuGet `Microsoft.AspNetCore.Authentication.JwtBearer`

В файле appsettings.json добавьте секцию с данными сервера аутентификации:

```
"AuthServer": {
  "Host": "http://localhost:8080",
  "Realm": "[Имя вашего Realm]"
}
```

В папке Models создайте класс, описывающий данные сервера аутентификации:

```
internal class AuthServerData
    {
        public string Host { get; set; }
        public string Realm { get; set; }
    }
```

В классе Program зарегистрируйте сервис аутентификации:

```
var authServer = builder.Configuration
    .GetSection("AuthServer")
    .Get<AuthServerData>();

// Добавить сервис аутентификации
builder.Services.AddAuthentication(JwtBearerDefaults.AuthenticationScheme)
    .AddJwtBearer(JwtBearerDefaults.AuthenticationScheme, o =>
    {
        // Адрес метаданных конфигурации OpenID
        o.MetadataAddress = $"{authServer.Host}/realms/{authServer.Realm}/.well-
known/openid-configuration";

        // Authority сервера аутентификации
        o.Authority = $"{authServer.Host}/realms/{authServer.Realm}";

        // Audience для токена JWT
        o.Audience = "account";

        // Запретить HTTPS для использования локальной версии Keycloak
        // В рабочем проекте должно быть true
```

```
        o.RequireHttpsMetadata = false;
    });
```

Опишите политику авторизации «admin», требующую наличие Claim с именем «role» и значением «POWER-USER».

```
builder.Services.AddAuthorization(opt =>
{
    opt.AddPolicy("admin", p => p.RequireRole("POWER-USER"));
});
```

В классе Program добавьте использование middleware аутентификации.

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

В регистрации конечных точек API для объектов (не категорий) разрешите неавторизованный доступ только на чтение данных. Для остальных запросов используйте созданную политику «admin»:

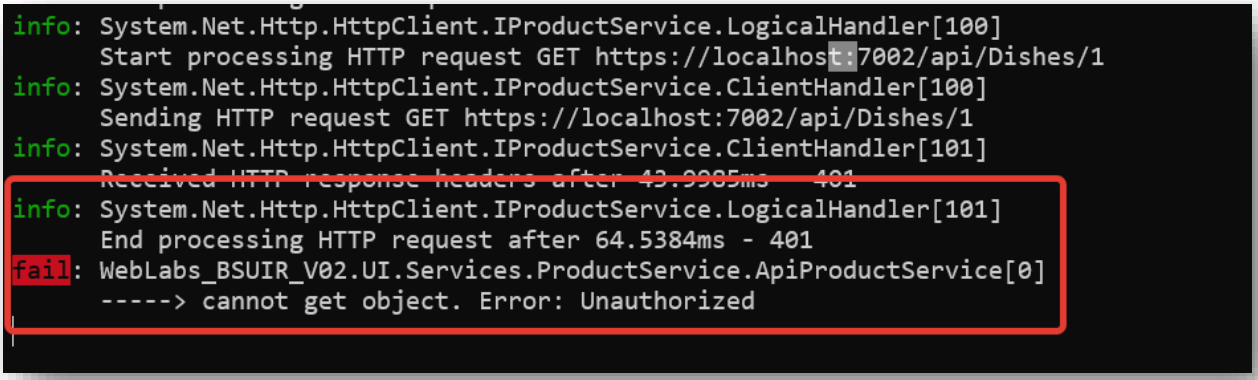
```
var group = routes.MapGroup("/api/Dish")
                  .WithTags(nameof(Dish))
                  .DisableAntiforgery()
                  .RequireAuthorization("admin");

 group.MapGet("/{category:alpha?}", async (IMediator mediator,
                                           string? category,
                                           int pageNo=1) =>
 {
     . . .
 })
                .WithName("GetAllDishes")
                .WithOpenApi()
                .AllowAnonymous();
```

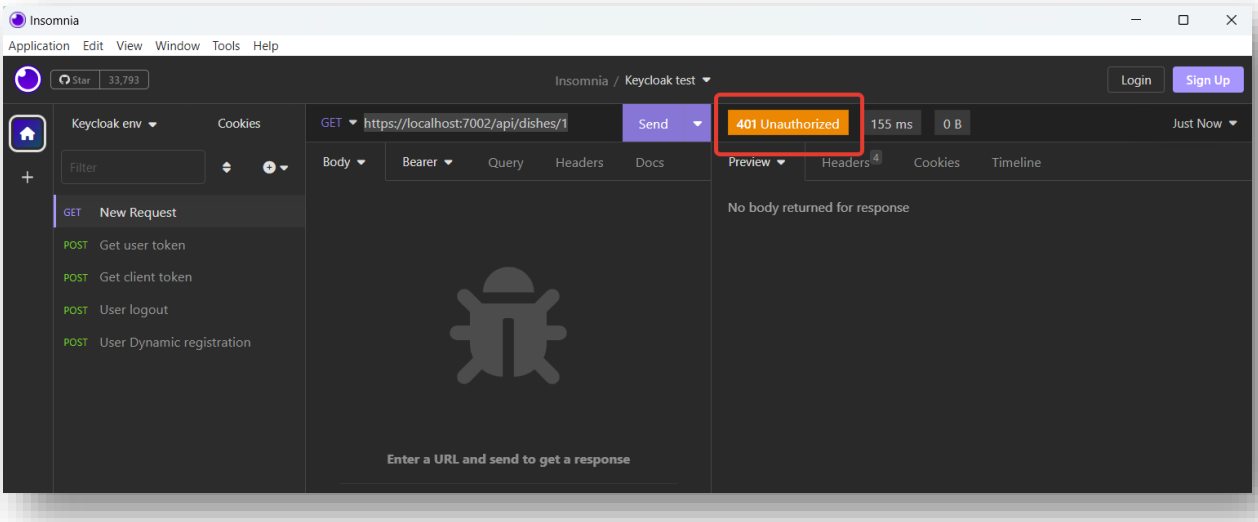
4.6.1. Проверка авторизации API

Запустите проект. Убедитесь, что в режиме администратора не выполняются действия, связанные с изменением данных.

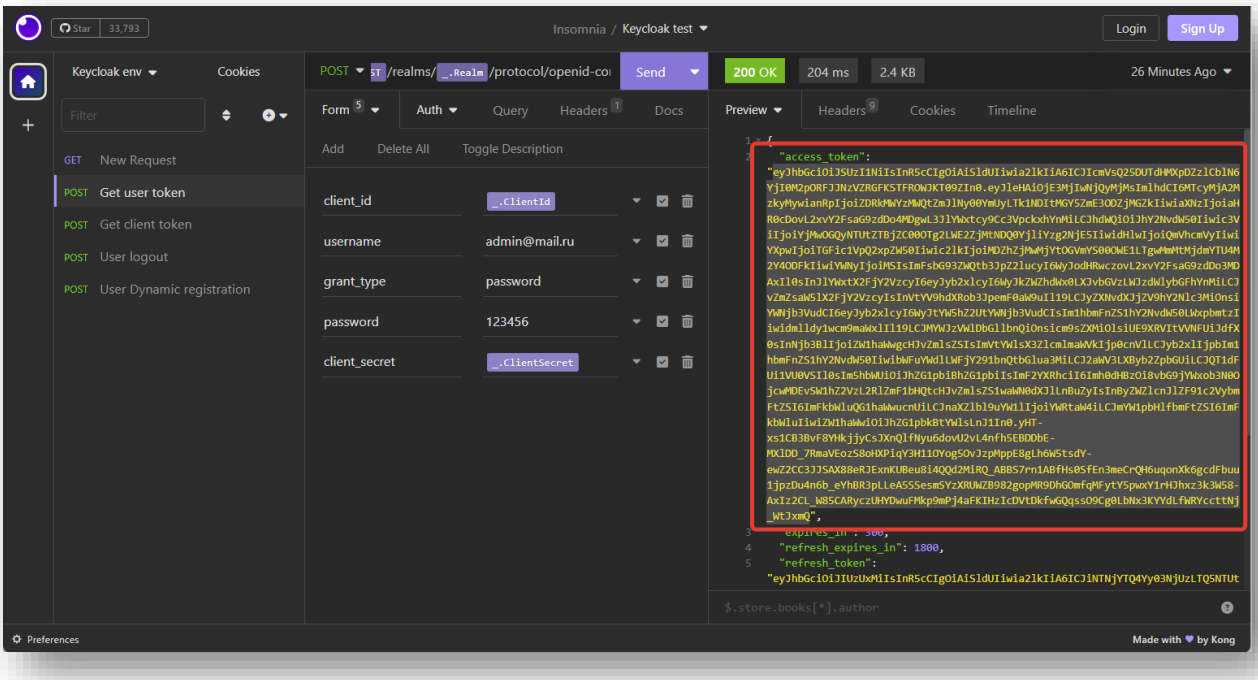
В консоли приложения найдите запись о получении кода 401 (Unauthorized)



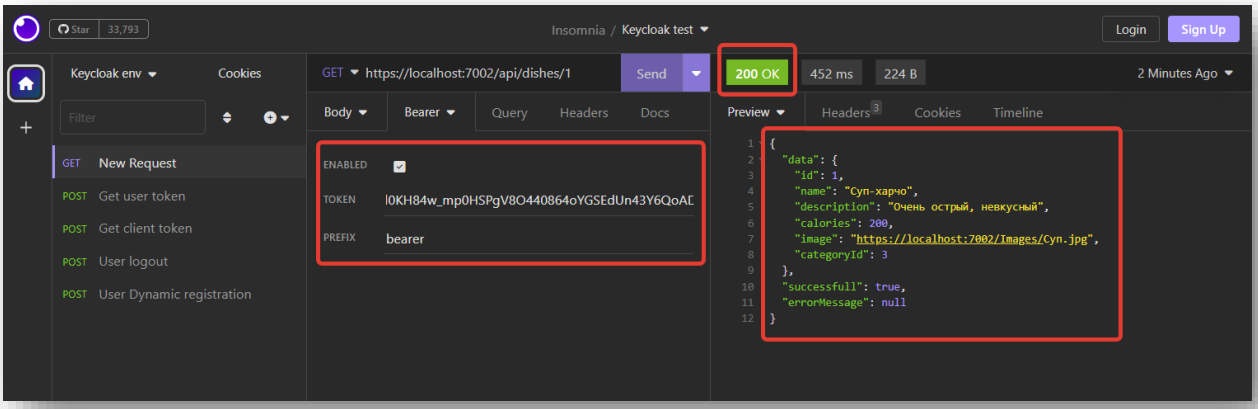
В Postman или Insomnia выполните запрос к API для получения одного объекта по ID (пример запроса: https://localhost:7002/api/dishes/1). Должен быть получен код 401 (Unauthorized). Ниже приведены примеры при работе с Insomnia:



Отправьте запрос для получения токена пользователя, у которого назначена роль POWER-USER. Скопируйте «access-token».



В запросе к API для получения одного объекта по ID добавьте заголовок авторизации типа «bearer», вставьте скопированный access-token. Отправьте запрос. Должен быть получен запрошенный объект:



4.7. Настройка проекта XXX.UI

Добавьте в проект пакеты NuGet

Microsoft.AspNetCore.Authentication.OpenIdConnect

Microsoft.AspNetCore.Authentication.JwtBearer

В файле appsettings.json добавьте секцию с данными для подключения к Keycloak:

```
"Keycloak": {
  "Host": "http://localhost:8080",
  "Realm": [Имя вашего Realm],
  "ClientId": [Id вашего клиента],
  "ClientSecret": [Client secret вашего клиента]
}
```

В папке HelperClasses опишите класс, инкапсулирующий данные для подключения к Keycloak:

```
internal class KeycloakData
{
    public string Host { get; set; }
    public string Realm { get; set; }
    public string ClientId { get; set; }
    public string ClientSecret { get; set; }
}
```

В классе HostingExtensions зарегистрируйте конфигурацию Keycloak:

```
builder.Services
      .Configure<KeycloakData>(builder.Configuration.GetSection("Keycloak"));
```

В классе Program добавьте аутентификацию Cookie и OpenIdConnect.

```
builder.Services
    .AddAuthentication(options =>
    {
        options.DefaultScheme = CookieAuthenticationDefaults.AuthenticationScheme;
        options.DefaultChallengeScheme = "keycloak";
    })
    .AddCookie()
    .AddOpenIdConnect("keycloak", options =>
    {
        options.Authority = $"{keycloakData.Host}/auth/realms/{keycloakData.Realm}";
        options.ClientId = keycloakData.ClientId;
        options.ClientSecret = keycloakData.ClientSecret;
        options.ResponseType = OpenIdConnectResponseType.Code;
        options.Scope.Add("openid"); // Customize scopes as needed
        options.SaveTokens = true;
        options.RequireHttpsMetadata = false; // позволяет обращаться к локальному
Keycloak по http
        options.MetadataAddress =
$"{keycloakData.Host}/realms/{keycloakData.Realm}/.well-known/openid-configuration";
    });
```

Добавьте политику авторизации (аналогично, как в проекте XXX.API)

```
// Добавить политику авторизации
builder.Services.AddAuthorization(opt =>
        opt.AddPolicy("admin", p => p.RequireRole("POWER-USER")));
```

Добавьте ограничение доступа к страницам (Razor pages) только для роли “admin”:

```
app.MapRazorPages()
        .RequireAuthorization("admin");
```

Запустите проект. Убедитесь, что при переходе на страницу «Администрирование» выполняется переход на страницу Login сервера Keycloak. При этом данные в API не сохраняются и в консоли приложения выводится сообщение об отсутствии прав доступа.

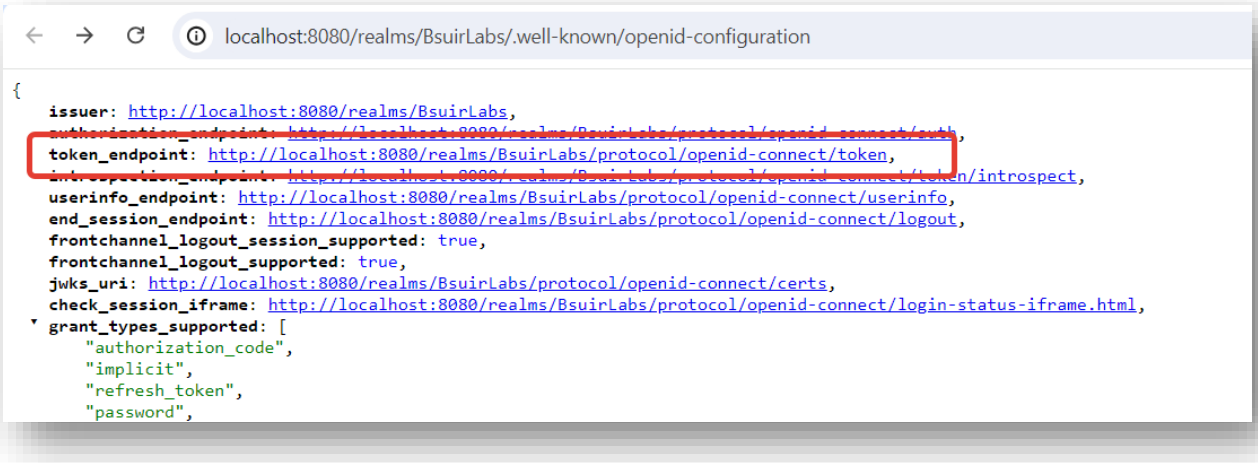
4.7.1. Получение токена аутентификации

Для того, чтобы получить доступ к API, необходимо передать токен аутентификации (access-token). Нам понадобятся два токена:

Токен приложения - получается приложением при обращении к конечной точке токена на сервере Keycloak. Он понадобится для регистрации пользователей.

Токен пользователя – получается при авторизации пользователя в системе. Он понадобится для обращения к API.

Конечную точку получения токена можно найти на странице свойств Realm. Пример конечной точки:



Для получения токена создайте сервис. В папку Services добавьте папку Authentication.

Опишите интерфейс ITokenAccessor:

```
public interface ITokenAccessor
{
    /// <summary>
    /// Добавление заголовка Authorizition : bearer
    /// </summary>
    /// <param name="httpClient">HttpLient, в который добавляется
заголовок</param>
    /// <param name="isClient">если true – получить токен клиента;
сли false – получить токен пользователя</param>
    /// <returns></returns>
    Task SetAuthorizationHeaderAsync(HttpClient httpClient,
                                     bool isClient);
}
```

Опишите класс KeycloakTokenAccessor, реализующий интерфейс ITokenAccessor.

При реализации метода GetTokenAsync нужно учитывать следующее:

Если пользователь вошел в систему, то токен можно извлечь из объекта HttpContext. Для этого нужно:

– в классе Program зарегистрировать сервис **HttpContextAccessor**:

```
builder.Services.AddHttpContextAccessor();
```

– в конструкторе класса KeycloakAuthService получить контекст:

```
_httpContext = httpContextAccessor.HttpContext;
```

Теперь можно получить токен:

```
var token = await _httpContext.GetTokenAsync("access_token");
```

Если пользователь не входил в систему, то нужно получить токен клиента. Для этого в конструктор класса KeycloakAuthService нужно внедрить данные сервера Keycloak (зарегистрированы как IOptions<KeycloakData>) и HttpClient.

Зарегистрируйте созданный сервис в классе HostingExtensions:

```
builder.Services.AddHttpClient<ITokenAccessor, KeycloakTokenAccessor>();
```

Пример реализации:

```
public class KeycloakTokenAccessor(
    IHttpContextAccessor contextAccessor,
    IOptions<KeycloakData> options,
    HttpClient httpClient) : ITokenAccessor
{
    public async Task SetAuthorizationHeaderAsync(HttpClient httpClient,
```

```
                                                    bool isClient)
    {
        string token = isClient
            ? await GetClientToken()
            : await GetUserToken();

        httpClient
            .DefaultRequestHeaders
            .Authorization = new AuthenticationHeaderValue("bearer", token);

    }

    async Task<string> GetUserToken()
    {
        var context = contextAccessor.HttpContext;
        var authSession = await context.AuthenticateAsync("keycloak");
        if (authSession?.Principal == null)
        {
            throw new AuthenticationFailureException("Пользователь
неавторизован");
        }
        return await context.GetTokenAsync("keycloak", "access_token");
    }

    async Task<string> GetClientToken()
    {
        // Keycloak token endpoint
        var requestUri =
$"{options.Value.Host}/realms/{options.Value.Realm}/protocol/openid-
connect/token";

        // Http request content
        HttpContent content = new FormUrlEncodedContent([
            new KeyValuePair<string,string>
                        ("client_id",options.Value.ClientId),
            new KeyValuePair<string,string>
                        ("grant_type","client_credentials"),
            new KeyValuePair<string,string>
                        ("client_secret",options.Value.ClientSecret)
            ]);

        // send request
        var response = await httpClient.PostAsync(requestUri, content);

        if (!response.IsSuccessStatusCode)
        {
            throw new HttpRequestException(response.StatusCode.ToString());
        }

        // extract access token from response
        var jsonString = await response.Content.ReadAsStringAsync();
        return JsonObject.Parse(jsonString)["access_token"].GetValue<string>();
    }
}
```

4.7.2. Использование токена в сервисе ApiProductService

Внедрите сервис ITokenAccessor в конструктор класса ApiProductService.

Во всех методах класса ApiProductService перед отправкой запроса http вызывайте метод `SetAuthorizationHeaderAsync` внедренного сервиса:

```
try
 {
     await _tokenAccessor.SetAuthorizationHeaderAsync(_httpClient, false);
 }
 catch(Exception e)
 {
     return ResponseData<Dish>
     .Error($"Объект не добавлен. Error: {e.Message}");
 }
```

4.7.3. Проверка работы

Запустите проект. Убедитесь, что в режиме администратора данные модифицируются.

4.8. Регистрация на сервере аутентификации

Задача: создать страницу регистрации нового пользователя.

– Предусмотреть сохранение аватара пользователя.

– Аватар сохранять в папке wwwroot/Images проекта XXX.UI.

– Имя файла аватара назначать случайным образом.

– Url файла аватара сохранять в атрибуте «Avatar» пользователя.

– Если аватар отсутствует, в атрибут «Avatar» записать Url стандартной картинки анонимного пользователя.

4.8.1. Общая информация

Сервер Keycloak предоставляет страницу регистрации. Но ее функционал не предусматривает сохранение файла изображения.

Создадим свою страницу регистрации, а для сохранения данных нового пользователя воспользуемся Admin API сервера Keycloak (см. https://www.keycloak.org/docs-api/latest/rest-api/index.html#_users)

4.8.2. Сервис сохранения файлов

В папке Services/FileService опишите интерфейс файлового сервиса:

```
public interface IFileService
```

```
{
    /// <summary>
    /// Сохраняет данные в файле
    /// </summary>
    /// <param name="file">данные для сохранения</param>
    /// <returns>адрес сохраненного файла</returns>
    Task<string> SaveFileAsync(IFormFile file);
}
```

Опишите класс, реализующий интерфейс IFileService, сохраняющий файлы локально:

```
public class LocalFileService(IWebHostEnvironment environment)
                                                    : IFileService
{
    public async Task<string> SaveFileAsync(IFormFile file)
    {
        // сгенерировать имя файла
        var fileName =
            . . .

        // полчить путь к сохраняемому файлу
        var filePath =
            . . .

        // скопировать файл в поток
            . . .

        return Path.Combine("Images", fileName);
    }
}
```

4.8.3. Создание контроллера Account

Для регистрации пользователя в папке Models опишите класс, инкапсулирующий данные, необходимые для регистрации нового пользователя:

```
internal class RegisterUserViewModel
{
    [Required]
    public string Email { get; set; }
    [Required]
    public string Password { get; set; }
    [Required]
    [Compare(nameof(Password))]
    public string ConfirmPassword { get; set; }
    public IFormFile? Avatar { get; set; }
}
```

Добавьте в проект пустой контроллер MVC с именем AccountController.

Опишите методы (Action) для регистрации нового пользователя. В качестве модели представления используйте класс RegisterUserViewModel. Требуется два метода. Первый – по методу Get отправляет клиенту форму для ввода регистрационных данных. Второй – по методу Post принимает данные и регистрирует пользователя на сервере аутентификации.

Для регистрации пользователя в Keycloak необходимо выполнить следующие шаги:

1) Получить JWT access-token-токен клиента.
2) Послать запрос на API администратора (https://www.keycloak.org/docs-api/latest/rest-api/index.html#_users). Конечная точка для регистрации пользователя: POST /admin/realms/{realm}/users.

В запросе установить заголовок Authorization: bearer { access-token }

С запросом передать StringContent, который содержит данные о новом пользователе в формате JSON (https://www.keycloak.org/docs-api/latest/rest-api/index.html#UserRepresentation). Пример данных о новом пользователе:

```
var userData = """
{
      "attributes": {
      "avatar" : "images/default-profile-picture.png"
      },
      "username": "igor@example.com",
      "email": "igor@example.com",
      "enabled": true,
      "emailVerified":true,
      "credentials": [{
            "temporary":false,
            "type": "password",
            "value": "123456"
            }]
      }
""";
```

Данные о новом пользователе можно сформировать, используя механизм интерполяции строк C#.

В приведенном ниже примере будет использоваться другой подход - сериализация объектов класса в JSON. Для этого опишите классы, инкапсулирующие данные о пользователе и данные Credentials. Поскольку эти классы не будут нигде больше использоваться, можно описать их в файле класса контроллера:

```
class CreateUserModel
{
    public Dictionary<string, string> Attributes { get; set; } = new();
    public string Username { get; set; }
    public string Email { get; set; }
    public bool Enabled { get; set; } = true;
    public bool EmailVerified { get; set; } = true;
    public List<UserCredentials> Credentials { get; set; } = new();
}

class UserCredentials
{
    public string Type { get; set; } = "password";
    public bool Temporary { get; set; } = false;
    public string Value { get; set; }
}
```

Для отправки запросов к Keycloak внедрите в конструктор класса контроллера объекты HttpClient и IOptions<KeycloakData>.

Для сохранения файла аватара внедрите IFileService.

Пример реализации:

```
public class AccountController(IHttpContextAccessor contextAccessor,
                               HttpClient httpClient,
                               ITokenAccessor tokenAccessor,
                               IOptions<KeycloakData> options,
                               IFileService fileService
                                ) : Controller
{


    public IActionResult Register()
    {
        return View(new RegisterUserViewModel());
    }

    [HttpPost]
    [AutoValidateAntiforgeryToken]
    public async Task<IActionResult> Register(RegisterUserViewModel user)
    {
        if (ModelState.IsValid)
        {
            if (user == null)
```

```
            {
                return BadRequest();
            }

            try
            {
                await tokenAccessor.SetAuthorizationHeaderAsync(httpClient, true);
            }
            catch (Exception ex)
            {
                return Unauthorized();
            }

            var avatarUrl = "/images/default-profile-picture.png";
            // сохранить Avatar, если аватар был передан при регистрации
            if (user.Avatar != null)
            {
                avatarUrl = await fileService.SaveFileAsync(user.Avatar);
            }

            // Подготовка данных нового пользователя
            var newUser = new CreateUserModel();
            newUser.Attributes.Add("avatar", avatarUrl);
            newUser.Email = user.Email;
            newUser.Username = user.Email;
            newUser.Credentials.Add(new UserCredentials { Value = user.Password });

            // Keycloak user endpoint
            var requestUri =
$"{options.Value.Host}/admin/realms/{options.Value.Realm}/users";

            // Подготовить контент запроса
            var serializerOptions = new JsonSerializerOptions
            {
                PropertyNamingPolicy = JsonNamingPolicy.CamelCase
            };
            var userData = JsonSerializer.Serialize(newUser, serializerOptions);
            HttpContent content = new StringContent(userData, Encoding.UTF8,
"application/json");

            // Отправить запрос
            var response = await httpClient.PostAsync(requestUri, content);//,
serializerOptions);

            if (response.IsSuccessStatusCode)
            {
                return Redirect(Url.Action("Index", "Home"));
            }
            else return BadRequest(response.StatusCode);
        }
        return View(user);
    }


}
    class UserCredentials
    {
        public string Type { get; set; } = "password";
        public bool Temporary { get; set; } = false;
        public string Value { get; set; }
    }

    class CreateUserModel
    {
        public Dictionary<string, string> Attributes { get; set; } = new();
```

```
        public string Username { get; set; }
        public string Email { get; set; }
        public bool Enabled { get; set; } = true;
        public bool EmailVerified { get; set; } = true;
        public List<UserCredentials> Credentials { get; set; } = new();
    }
```

4.8.4. Представление Register

Создайте представление для Action Register контроллера Account. Воспользуйтесь scaffold (шаблонный элемент). Используйте шаблон Create. В качестве модели укажите класс `RegisterUserViewModel.`

Откройте созданное представление.

Добавьте в форму атрибут `enctype="multipart/form-data"`

Добавьте разметку для загрузки файла аватара

4.8.5. Проверка страницы регистрации

Запустите проект, введите адрес https://localhost:7002/account/register

Зарегистрируйте нового пользователя:

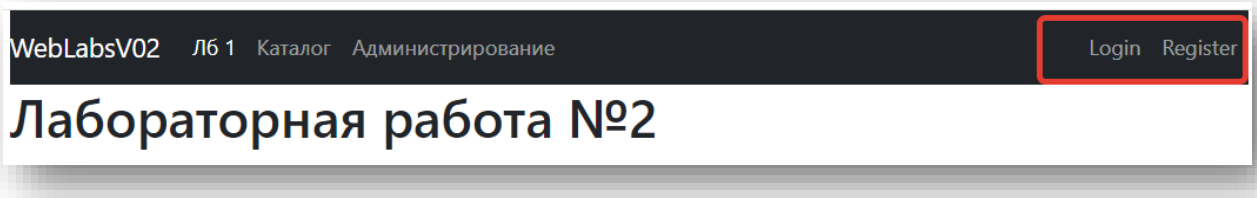
4.9. Доработка проекта XXX.UI

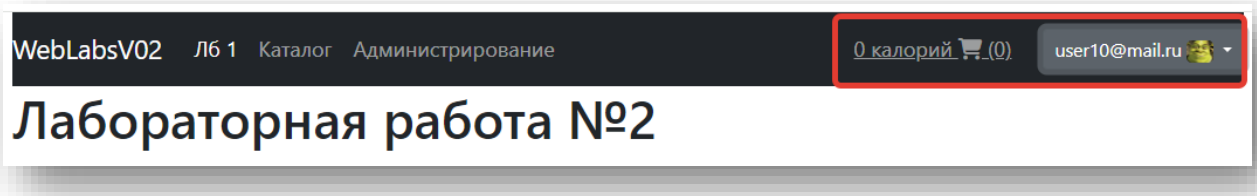
4.9.1. Задание

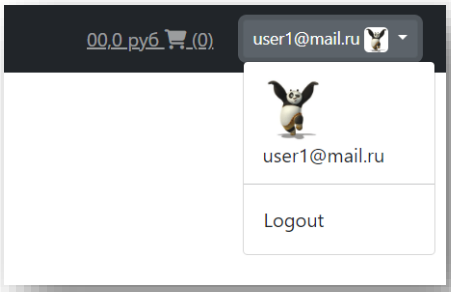
В основном проекте требуется реализовать следующее: информация, выводимая в меню «Информация пользователя», должна зависеть от того, вошел пользователь в систему, или нет. А именно:

- если пользователь не вошел в систему, показывать кнопки «Войти» («Login») и «Зарегистрироваться» («Register»).

- если пользователь вошел в систему, то показывать корзину, имя пользователя и аватар. Кнопка «Logout» должна работать.







4.9.2. Рекомендации к заданию

a) для реализации функций Login и Logout добавьте в контроллер Account два действия (action): Login, реализующее переадресацию на страницу входа сервера аутентификации, и Logout (по методу HttpPost), реализующий выход. Пример реализации:

```
public async Task Login()
{
    await HttpContext.ChallengeAsync(
        "keycloak",
        new AuthenticationProperties { RedirectUri = Url.Action("Index", "Home")
});
}
[HttpPost]
public async Task Logout()
{
    await
HttpContext.SignOutAsync(CookieAuthenticationDefaults.AuthenticationScheme);
    await HttpContext.SignOutAsync("keycloak",
        new AuthenticationProperties { RedirectUri = Url.Action("Index", "Home")
});
}
```

b) В частичном представлении _UserInfoPartial:

Для определения того, что пользователь аутентифицирован, можно использовать свойство User:

```
User.Identity.IsAuthenticated
```

Имя пользователя можно получить из соответствующего утверждения (Claim). Аватар можно получить из соответствующего утверждения (Claim):

```
@{
    var name = @User
        .Claims
        .FirstOrDefault(c => c.Type.Equals("preferred_username",
                            StringComparison.OrdinalIgnoreCase))?
        .Value;

    var avatar = @User
        .Claims
        .FirstOrDefault(c => c.Type.Equals("avatar",
                            StringComparison.OrdinalIgnoreCase))?
        .Value;
}
```

c) Кнопки Login и Logout должны адресовать на соответствующие методы контроллера Identity – см. п. a) (**использовать тэг-хелперы**)

Запустите проект. Проверьте возможность входа/выхода в систему, а также правильность работы меню «Информация пользователя».

5. Контрольные вопросы

1. Какой механизм аутентификации имеет встроенную поддержку в ASP.Net Core?
2. Что описывают классы ClaimsPrincipal и ClaimsIdentity?
3. Как подключить Middleware аутентификации и авторизации?
4. Приведите пример использования свойства HttpContext.User.
5. Как в коде проверить, что пользователь прошел аутентификацию?
6. Как получить значение Claim пользователя?
7. Как получить Id пользователя, прошедшего аутентификацию?
8. Как разрешить доступ к контроллеру только для пользователей с ролью «manager»?
9. Как создать политику авторизации с помощью Claim?
10. Как создать куки аутентификации с помощью объекта HttpContext?
11. Как добавить в проект использование системы членства Microsoft.AspnetCore.Identity?
12. Как с помощью системы членства Microsoft.AspnetCore.Identity создать нового пользователя?
13. Как с помощью системы членства Microsoft.AspnetCore.Identity осуществить вход пользователя в систему?
14. Как с помощью системы членства Microsoft.AspnetCore.Identity добавить Claim пользователю?
15. Какой интерфейс используется в Microsoft.AspnetCore.Identity для доступа к хранилищу пользователей?