

## **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)**

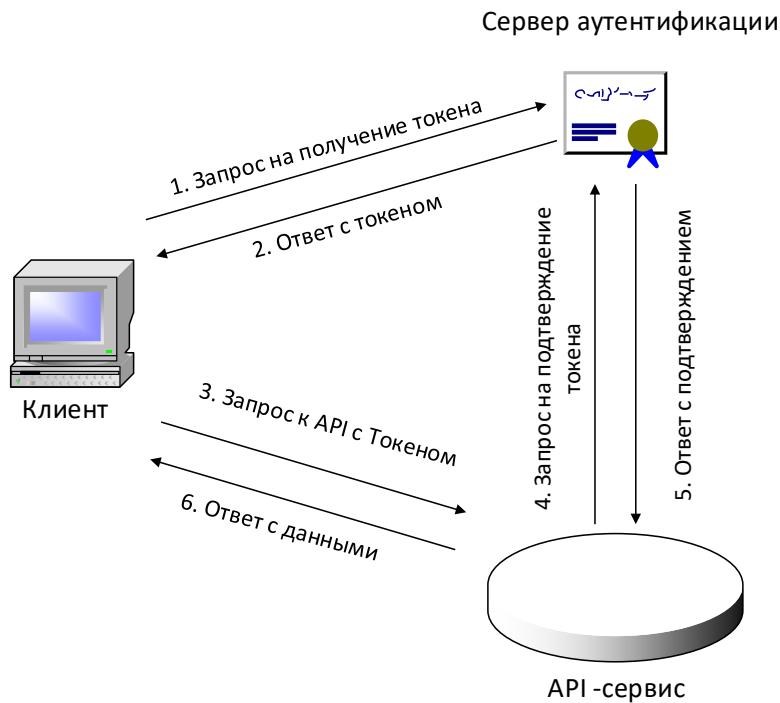
#### **3. Предварительная информация**

##### **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

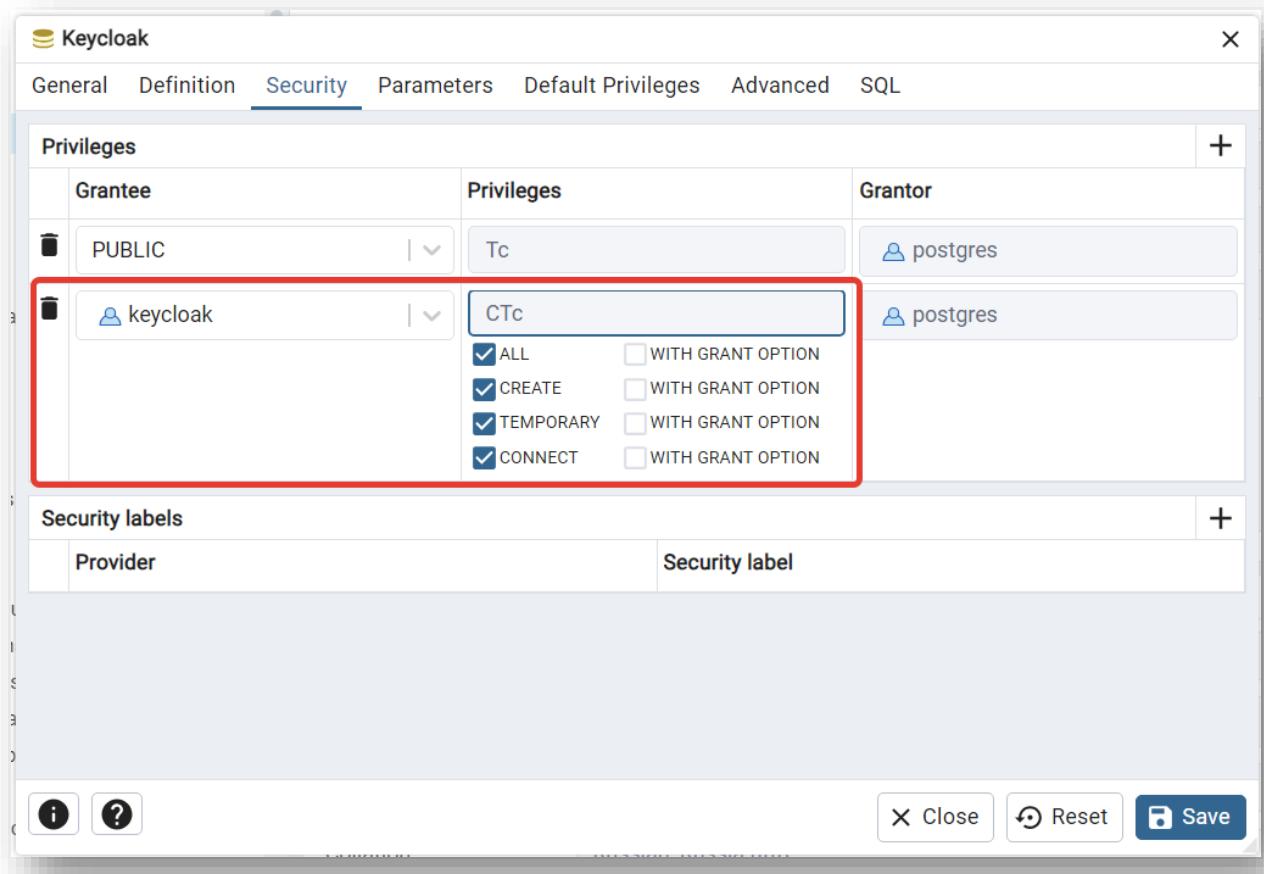
#### 4.2.2. Подключение базы данных к Keycloak

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 Имя пользователя

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

# 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 Имя базы данных (с учетом регистра)

```

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

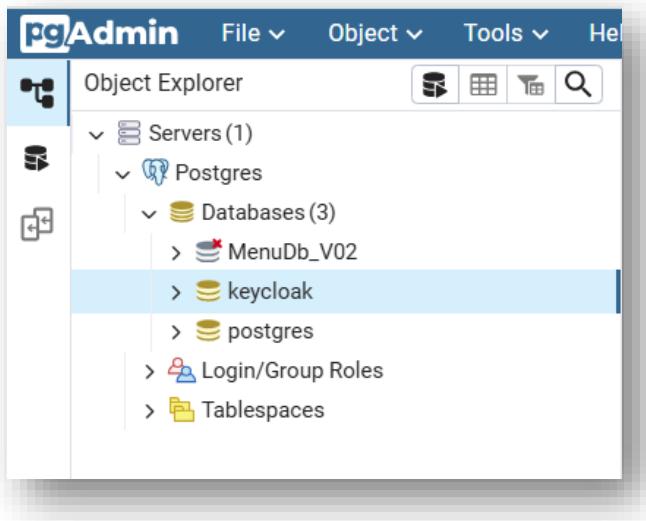
KEYCLOAK\_ADMIN=admin

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, указанным в переменной POSTGRES\_DB (в примере – keycloak).



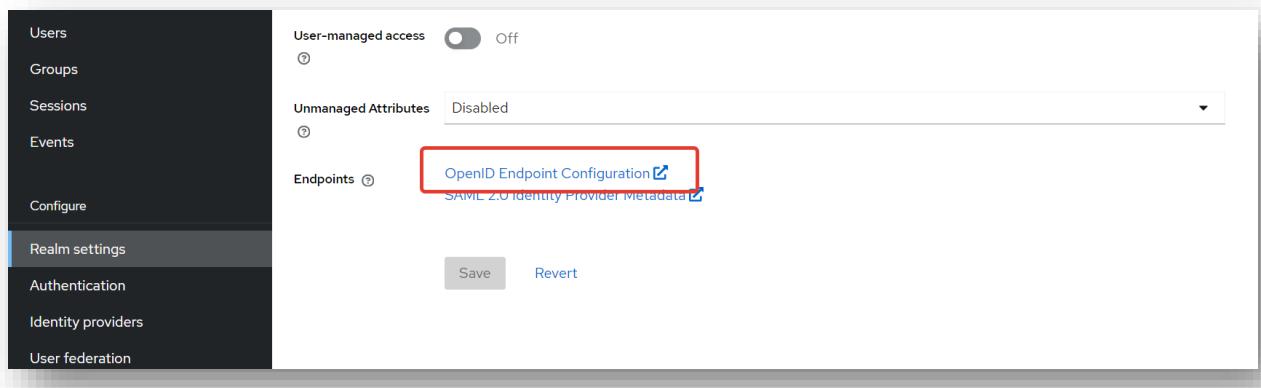
## 4.4. Настройка Keycloak

### 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:

The screenshot shows the Keycloak Admin UI with the 'TestRealm' selected in the dropdown. The left sidebar has 'Realm settings' selected. The top navigation bar includes 'General', 'Login' (which is active), 'Email', 'Themes', 'Keys', 'Events', 'Localization', 'Security defenses', and 'Sessions'. The main content area is titled 'Login screen customization' and contains three toggle switches: 'User registration' (Off), 'Forgot password' (Off), and 'Remember me' (Off). Below this is a section titled 'Email settings' with two toggle switches: 'Email as username' (On) and 'Login with email' (On), which are both highlighted with a red box. At the bottom is a switch for 'Duplicate emails' (Off).

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.

The screenshot shows the Keycloak Admin UI with the 'User profile' tab selected. The left sidebar has 'Realm settings' selected. The top navigation bar includes 'Keys', 'Events', 'Localization', 'Security defenses', 'Sessions', 'Tokens', 'Client policies', 'User profile' (which is active), and 'User registration'. The main content area is titled 'Attributes' and shows a table with three rows: 'username' (display name \${username}), 'email' (display name \${email}), and 'firstName' (display name \${firstName}). Above the table is a button labeled 'Create attribute' with a red box around it. There is also a dropdown menu for 'All groups'.

Attribute [Name] \* ⓘ avatar

Display name ⓘ \${avatar}

Multivalued ⓘ Off

Attribute group ⓘ None

Enabled when ⓘ  Always  Scopes are requested

**Required field** ⓘ  Off

**Permission**

Who can edit? ⓘ  User  Admin

Who can view? ⓘ  User  Admin

**Save** **Cancel**

#### 4.4.2. Создание клиента.

На вкладке Clients создайте клиента типа OpenID Connect.

Используйте название [Ваша фамилия]UiClient

Clients > Create client

**Create client**

Clients are applications and services that can request authentication of a user.

**General settings**

Client type ⓘ OpenID Connect

Client ID \* ⓘ LabsUiClient

Name ⓘ

Description ⓘ

Always display in UI ⓘ  Off

Разрешите Client Authentication, Authorization и Implicit Flow:

Clients > Create client

## Create client

Clients are applications and services that can request authentication of a user.

1 General settings  
2 Capability config  
3 Login settings

**Client authentication** On

**Authorization** On

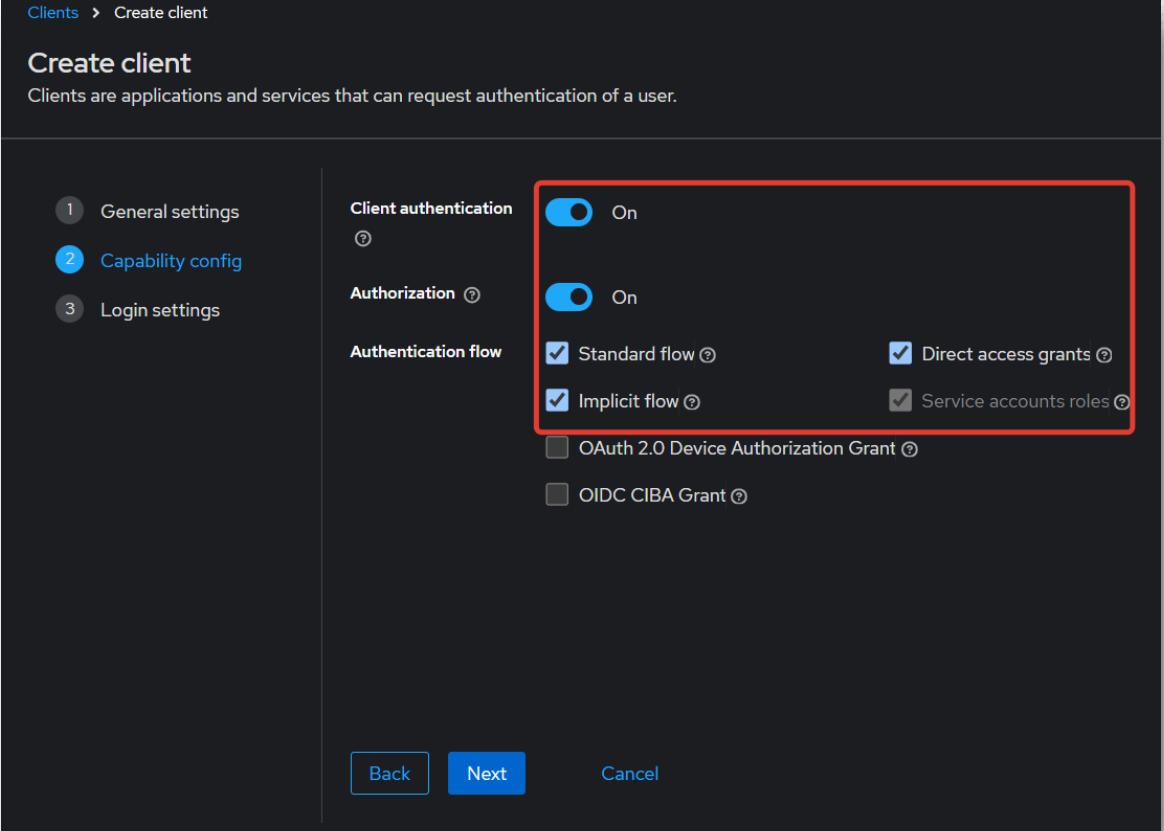
**Authentication flow**

- Standard flow
- Implicit flow
- OAuth 2.0 Device Authorization Grant
- OIDC CIBA Grant

Direct access grants

Service accounts roles

Back Next Cancel



Укажите Url вашего приложения:

Clients > Create client

## Create client

Clients are applications and services that can request authentication of a user.

1 General settings  
2 Capability config  
3 Login settings

**Root URL** https://localhost:7001

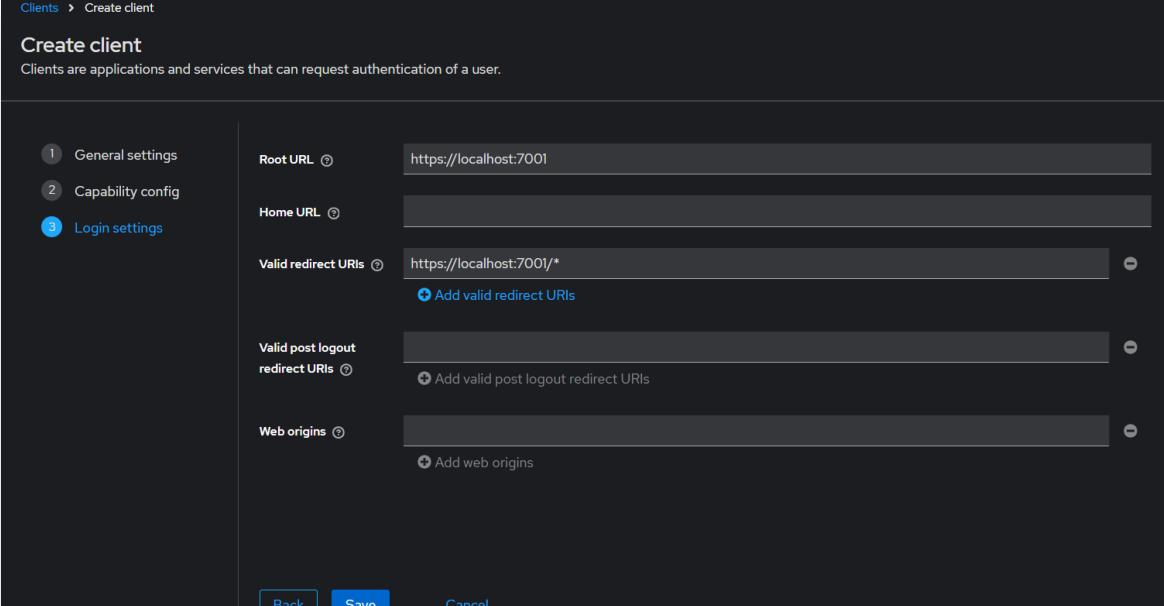
**Home URL**

**Valid redirect URIs** https://localhost:7001/\*  
[Add valid redirect URIs](#)

**Valid post logout redirect URIs**

**Web origins**

Back Save Cancel



На вкладке Credentials найдите поле Client Secret

The screenshot shows the Keycloak 'Clients' configuration interface. On the left sidebar, 'Clients' is selected. The top navigation bar has tabs: Settings, Keys, **Credentials**, Roles, Client scopes, Sessions, and Advanced. The 'Credentials' tab is active. Below the tabs, there's a section for 'Client Authenticator' with a dropdown set to 'Client Id and Secret'. A 'Save' button is below this. At the bottom, there's a 'Client Secret' field containing a long string of dots (...), with three icons (copy, eye, trash) next to it and a 'Regenerate' button on the right.

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

The screenshot shows the 'Client details' page for 'LabsUiClient'. The left sidebar shows 'Clients' selected. The top navigation bar includes 'Clients > Client details' and tabs: Settings, Keys, Credentials, Roles, Client scopes, Authorization, **Service accounts roles**, Sessions, and Advanced. The 'Service accounts roles' tab is active. A note says: 'To manage detail and group mappings, click on the username service-account-labsuiclient'. Below is a search bar, a 'Hide inherited roles' checkbox (which is checked), and an 'Assign role' button. A table lists roles: 'realm-management manage-users' (highlighted with a red box and a red arrow pointing to it), 'LabsUiClient uma\_protection', and 'default-roles-bsuirlabs\_v03'. The columns are 'Name', 'Inherited', and 'Description'.

#### 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

| Name              | From predefined mappers | Type         |
|-------------------|-------------------------|--------------|
| Client ID         | By configuration        | User Session |
| Client IP Address | Token mapper            | User Session |
| Client Host       | Token mapper            | User Session |

Выберите в списке User Attribute

| Choose any of the mappings from this table | Description   |
|--|---|
| Nonce backwards compatible                 | Adds the nonce claim to Access, Refresh and ID token  |
| Organization Membership                    | Map user Organization membership  |
| Pairwise subject identifier                | Calculates a pairwise subject identifier using a salted sha-256 hash and adds it to the 'sub' claim. See OpenID Connect specification for more info about pairwise subject identifiers. |
| Role Name Mapper                           | Map an assigned role to a new name or position in the token.  |
| Session State (session_state)              | Add Session State (session_state) claim   |
| Subject (sub)                              | Add Subject (sub) claim   |
| User Address                               | Maps user address attributes (street, locality, region, postal_code, and country) to the OpenID Connect 'address' claim.  |
| <b>User Attribute</b>                      | <b>Map a custom user attribute to a token claim.</b>  |
| User Client Role                           | Map a user client role to a token claim.  |
| User Property                              | Map a built in user property (email, firstName, lastName)   |

Выберите атрибут avatar

Clients > Client details > Dedicated scopes > Mapper details

User Attribute  
20c12392-66f1-4f99-8227-10a9f11a02d3

|                     |  |
|---------------------|--|
| Mapper type         | User Attribute                         |
| Name *              | avatar                                 |
| User Attribute      | avatar                                 |
| Token Claim Name    | avatar                                 |
| Claim JSON Type     | String                                 |
| Add to ID token     | <input checked="" type="checkbox"/> On |
| Add to access token | <input checked="" type="checkbox"/> On |

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:

Clients > Client details > Role details

POWER-USER

|             |                       |
|-------------|-----------------------|
| Role name   | POWER-USER            |
| Description | Администратор клиента |

Save Cancel

#### 4.4.5. Передача роли в токене в виде Claim

На вкладке «Client scopes» выберите «roles»

| Name            | Assigned type | Protocol       | Display order | Description  |
|-----------------|---------------|----------------|---------------|--|
| acr             | Default       | OpenID Connect | -             | OpenID Connect scope for add acr (authentication context)  |
| address         | Optional      | OpenID Connect | -             | OpenID Connect built-in scope: address                     |
| basic           | Default       | OpenID Connect | -             | OpenID Connect scope for add all basic claims to the token |
| email           | Default       | OpenID Connect | -             | OpenID Connect built-in scope: email                       |
| micropointe-jwt | Optional      | OpenID Connect | -             | Micropointe - JWT built-in scope                           |
| offline_access  | Optional      | OpenID Connect | -             | OpenID Connect built-in scope: offline_access              |
| phone           | Optional      | OpenID Connect | -             | OpenID Connect built-in scope: phone                       |
| profile         | Default       | OpenID Connect | -             | OpenID Connect built-in scope: profile                     |
| role_list       | Default       | SAML           | -             | SAML role list   |
| <b>roles</b>    | Default       | OpenID Connect | -             | OpenID Connect built-in scope: roles                       |

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

| Name             | Type             |
|------------------|------------------|
| audience resolve | Audience Resolve |
| User client role | Token mapper     |

- «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.

```

{
  "sub": "user0@mail.ru",
  "aud": "https://localhost:7001/resources",
  "exp": 1712955600,
  "iat": 1712952000,
  "nbf": 1712952000,
  "jti": "cm9maWxI1I19LCJMYWJzVWlDbG1lbnQiOnsicm9sZXMiolsiUE9XRVItVVFNUiJdfX0sInNjb3B1IjoiZW1haWwgchJvZmlsZSIsImVtYVlsX3Z1cm1maWVkJpb0cnV1LCJyb2x1IjpBI1hbmFnZS1hY2NvdW50IiwbWFuYWDlLWFjY291bnQtbGlua3MilCJ2aWV3LXByb2ZpbGUlCJQT1dFUi1VU0VSl0sIm5hbWUiOjhZG1pb1BhZG1pb1sImF2YXRhcit6Imh0dHBz0i8vbG9jYWxob3N00jcwMDEvSW1hZ2VzL2R1ZmF1bHQtcHJvZmlsZS1waWN0dXJ1LnBuZyIsInByZWZlcnJ1ZF91c2VybmtZSI6ImFkbWluQG1haWwucnUiLCJnaZ1bl9uYW1IjoiYWRtaW4iLCJmY1pbH1fbmFtZSI6ImFkbWluIiwiZW1haWwiOjhZG1pbkBtYWlsLnJ1In0.yHt-xs1CB3BvF8YHkjyyCsJXnQ1fNyuu6dovU2vL4nfh5EBDDbE-MX1DD_7RmaVEozS8oHXPiqY3H110Yog50vJzpMpE8gLh6W5tsdy-ewZ2CC3JJSAX88eREExnKUBeu8i4QQd2MiRQ_ABBS7rn1ABfHs0SFEn3meCrQH6uqonXk6gcdFbu1jpzDu4n6b_eYhBR3pLLeA5SSesmSYzXRUWZB982gopMR9DhG0mfqMFytY5pwX1rHJhxz3k3W58-AxIz2CL_W85CARyczUHYDwuFMkp9mPj4aFKIHzi_cDVtDkfwGQqss09Cg0LbNx3KYYdLfWRYccttNj_WtJxmQ
}
  
```

#### 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
- password – пароль созданного пользователя
- client\_secret – client secret для созданного клиента

**Ваш ответ: Правильный адрес API для получения токена:**

```

{
  "access_token": "eyJhbGciOiJIUzI1NiIsInR5cC...",
  "token_type": "Bearer",
  "expires_in": 3600,
  "refresh_expires_in": 1800,
  "refresh_token": "eyJhbGciOiJIUzI1Ni...",
  "not-before-policy": 0,
  "scope": "read write"
}

```

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;
});
```

```
    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();
```

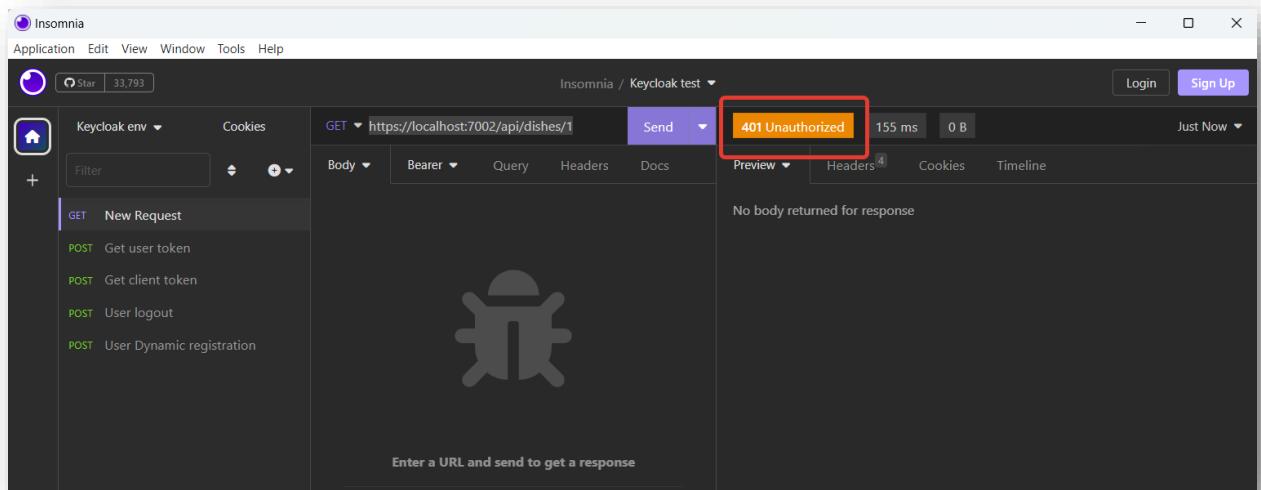
#### 4.6.1. Проверка авторизации API

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

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

```
info: System.Net.Http.HttpClient.IProductService.LogicalHandler[100]
  Start processing HTTP request GET https://localhost:7002/api/Dishes/1
info: System.Net.Http.HttpClient.IProductService.ClientHandler[100]
  Sending HTTP request GET https://localhost:7002/api/Dishes/1
info: System.Net.Http.HttpClient.IProductService.ClientHandler[101]
  Received HTTP response headers after 43.9985ms - 401
info: System.Net.Http.HttpClient.IProductService.LogicalHandler[101]
  End processing HTTP request after 64.5384ms - 401
fail: WebLabs_BSUIR_V02.UI.Services.ProductService.ApiProductService[0]
    -----> cannot get object. Error: 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».

Insomnia / Keycloak test

POST /realms/.well-known/protocol/openid-connect/token

200 OK | 204 ms | 2.4 KB | 26 Minutes Ago

Preview Headers Cookies Timeline

access\_token: eyJhbGciOiJSUzI1NiIsInR5Ct...  
refresh\_token: eyJhbGciOiJIUzI1Ni...  
refresh\_expires\_in: 1800

Expires in: 300

\$.store.books[\*].author

client\_id: .ClientId

username: admin@mail.ru

grant\_type: password

password: 123456

client\_secret: .ClientSecret

Form Auth Query Headers Docs

Add Delete All Toggle Description

GET New Request

POST Get user token

POST Get client token

POST User logout

POST User Dynamic registration

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:

The screenshot shows the Insomnia REST client interface. At the top, there's a navigation bar with a purple circular icon, a 'Star' button, and a '33,793' count. The title is 'Insomnia / Keycloak test'. On the right are 'Login' and 'Sign Up' buttons. Below the title, the URL is set to 'GET https://localhost:7002/api/dishes/1'. The status bar indicates a '200 OK' response, 452 ms duration, and 224 B size, with a timestamp of '2 Minutes Ago'. The main area has tabs for 'Body', 'Bearer', 'Query', 'Headers', and 'Docs'. The 'Body' tab is selected, showing the response body: { "data": { "id": 1, "name": "Суп-харчо", "description": "Очень острый, невкусный", "calories": 200, "image": "https://localhost:7002/Images/Cyn.jpg", "categoryId": 3 }, "successfull": true, "errorMessage": null } . The 'Headers' tab shows a 'Content-Type' header of 'application/json; charset=UTF-8'. The 'Cookies' and 'Timeline' tabs are also visible.

#### **4.7. Настройка проекта XXX.UI**

## Добавьте

E

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:

```
{
  issuer: "http://localhost:8080/realms/BuirLabs",
  authorization_endpoint: "http://localhost:8080/realms/BuirLabs/protocol/openid-connect/auth",
  token_endpoint: "http://localhost:8080/realms/BuirLabs/protocol/openid-connect/token",
  end_user_endpoint: "http://localhost:8080/realms/BuirLabs/protocol/openid-connect/end-user",
  userinfo_endpoint: "http://localhost:8080/realms/BuirLabs/protocol/openid-connect/userinfo",
  end_session_endpoint: "http://localhost:8080/realms/BuirLabs/protocol/openid-connect/logout",
  frontchannel_logout_supported: true,
  frontchannel_logout_supported: true,
  jwks_uri: "http://localhost:8080/realms/BuirLabs/protocol/openid-connect/certs",
  check_session_iframe: "http://localhost:8080/realms/BuirLabs/protocol/openid-connect/login-status-iframe.html",
  grant_types_supported: [
    "authorization_code",
    "implicit",
    "refresh_token",
    "password",
  ],
}
```

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

Опишите интерфейс ITokenAccessor:

```

public interface ITokenAccessor
{
    /// <summary>
    /// Добавление заголовка Authorization : bearer
    /// </summary>
    /// <param name="httpClient">HttpClient, в который добавляется
    /// заголовок</param>
    /// <param name="isClient">если true – получить токен клиента;
    /// если 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](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 =
            . . .

        // получить 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](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 Create UserModel
{
    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:

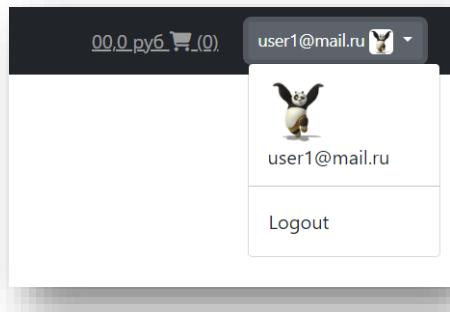
### 4.9. Доработка проекта XXX.UI

#### 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. а) (**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?