Создание кастомизированных логин-модулей

Версия 1.0.0

Задача:

Содержание

[1. Введение 3](#_Toc66891268)

[2. Регистрация логин модуля 3](#_Toc66891269)

[3. Изменения в мета-информации о приложении 3](#_Toc66891270)

[4. Принцип работы логин модуля 3](#_Toc66891271)

[5. Пример реализации 4](#_Toc66891272)

**Кастомизированные логин-модули**

# Введение

Для реализации дополнительных или расширенных процессов аутентификации (DFA, дополнительные проверки, авторизация через сторонние сервисы) потребуется добавить логин модуль согласно данной документации.

# Регистрация логин модуля

Зарегистрировать логин модуль через вызов пакетной функции или интерфейс. Добавить ему тип авторизации client\_credentials и роли OALoginModule,OAViewSession,OAEditSession.

Pkg\_Oauth.createClient(clientShortName => 'LoginModule',

clientName => 'LoginModule',

clientNameEn => 'LoginModule',

applicationType => 'web',

grantTypeList => 'client\_credentials',

roleShortNameList => 'OALoginModule,OAViewSession,OAEditSession',

loginModuleUri => null,

operatorId => 1);

# Изменения в мета-информации о приложении

В описание приложения в системе OAuth добавлено новое поле loginModuleUri. Оно содержит в себе URL стартовой точки аутентификации. Если значение поля null, то используется стандартная аутентификация AccessOperator.

Pkg\_Oauth.createClient(clientShortName => 'Client',

clientName => 'Client',

clientNameEn => 'Client',

applicationType => 'web',

grantTypeList => 'authorization\_code',

roleShortNameList => null,

loginModuleUri => '/login-module',

operatorId => 1);

# Принцип работы логин модуля

Предположим, что у нас существует зарегистрированное в OAuth приложение NFA:

Pkg\_Oauth.createClient(clientShortName   => ‘Client’,

                                  clientName        => 'Client',

                                  clientNameEn      => ‘Client En’,

                                  applicationType   => 'web',

                                  grantTypeList     => 'authorization\_code,refresh\_token',

                                  roleShortNameList => null,

                                  loginModuleUri    => '/login’,

                                  operatorId        => 1);

Pkg\_Oauth.createClientUri(clientShortName => ‘NFAClientId’,

                                     clientUri       => '/oauth-callback’,

                                     operatorId      => 1);

И логин-модуль:  
                Pkg\_Oauth.createClient(clientShortName   => ‘LoginModule’,

                                  clientName        => 'Login',

                                  clientNameEn      => ‘Login',

                                  applicationType   => 'web',

                                  grantTypeList     => 'client\_credentials',

                                  roleShortNameList => ’OALoginModule,OAViewSession,OAEditSession’,

                                  loginModuleUri    => null,

                                  operatorId        => 1);

Тогда можно следовать следующему алгоритму:

1. OAuth Authorization Endpoint (/authorize) инициирует авторизационную сессию и переводит пользователя на страницу логина, указанную в loginModuleUri в нашем случае /login?response\_type=code&client\_id=Client&redirect\_uri=/oauth-callback&state=state&session\_id=SID

Параметры, переданные на логин страницу, потребуются в дальнейшем для подтверждения аутентификации в OAuth (мета информация о сессии)

1. Пользователь проходит аутентификацию.
2. Если все проверки пройдены, то:
   1. Сервис логин модуля получает *clientSecret* по *clientId* вызовом следующей функции:

? := pkg\_OAuth.findClient(

        clientShortName => LoginModule,

        maxRowCount => 1,

        operatorId => 1

      );

end;

Возвращается курсор с описанием клиентского приложения, нужное поле CLIENT\_SECRET.

В JepRia есть утилитарный класс OAuthDBHelper, который реализует этот функционал.

Пункт 3.а можно пропустить, если указать client\_secret из базы явным образом, например, через переменную окружения Tomcat. Но это потребует конфигурации на всех окружениях. Потому что client\_secret везде будет разный.

* 1. Логин модуль обновляет сессию авторизации вызовом сервиса …/oauth/api/session/{sessionId} с мета-информацией о пользователе.

в HTTP заголовок Authorization нужно подставить значение: Basic base64(client\_id:client\_secret)

передается JSON вида:

{

    "state": "state",

    "operatorId": "integer",

    "username": "string",

    "responseType": "code",

    ''redirectUri'':”/oauth-callback”,

    ''clientId'': “Client”,

}

1. При успешном подтверждении авторизации OAuth вернет ответ с кодом 302 и в HTTP заголовок Location = /oauth-callback?code=authorization\_code&state=state
2. Нужно перевести пользователя на callback URL из заголовка Location
3. Клиентская часть запрашивает токен POST запросом к /oauth/api/token

# Пример реализации

Пример реализации можно посмотреть в Examples/simple-jsp-login-module.