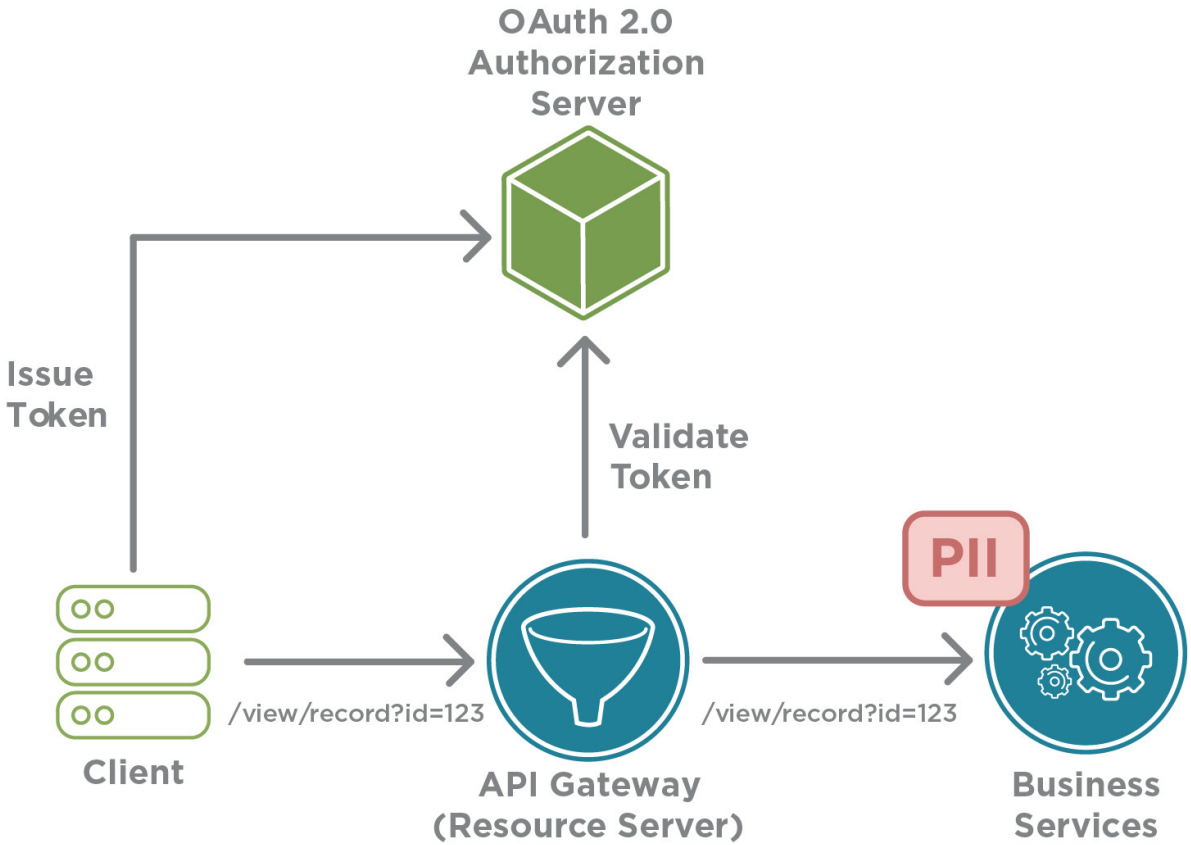
Методы аунтетификации OAuth + UMA Protocol flow

Pavel

Outline

OAuth 2.0

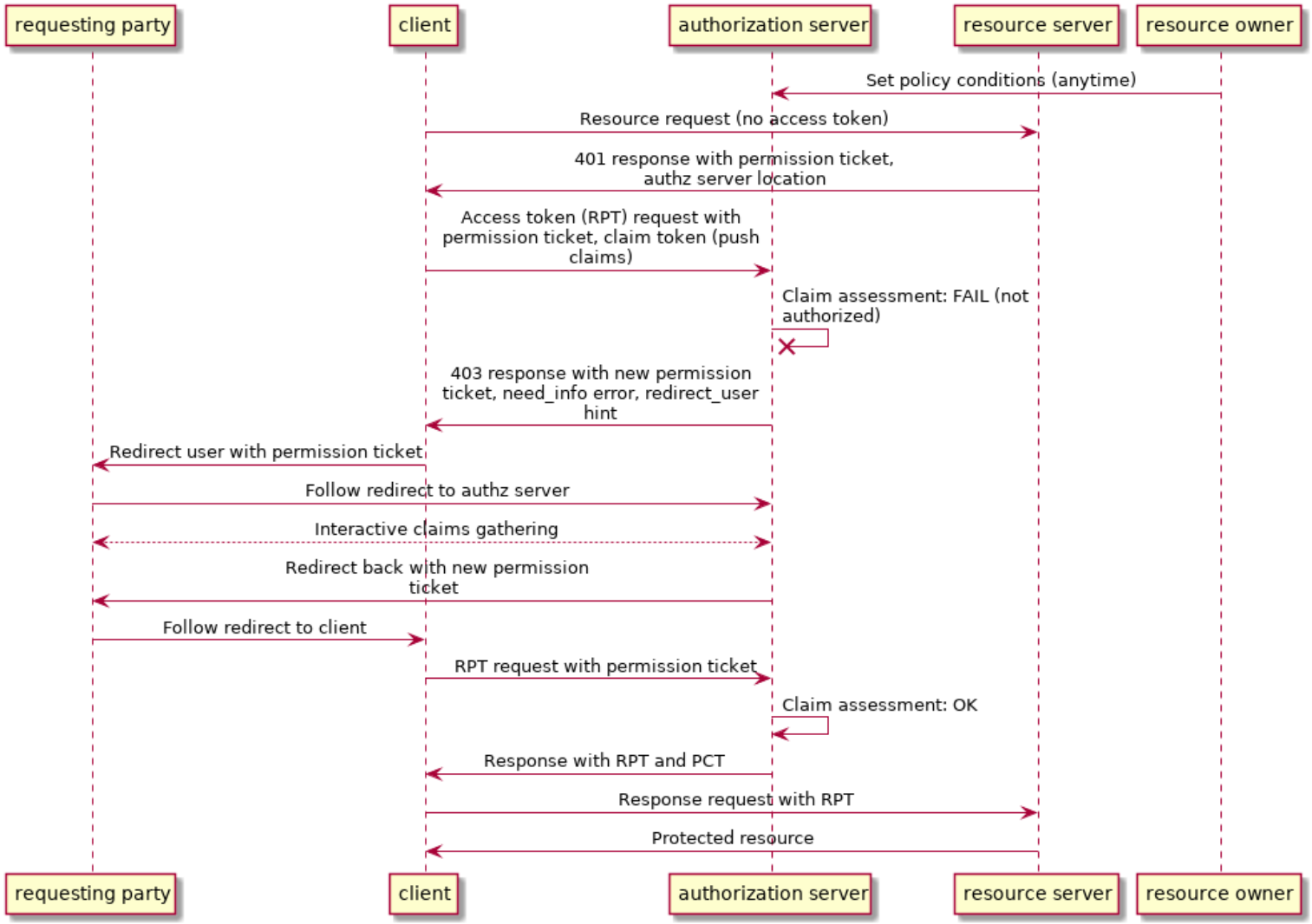

OAuth 2.0
Authorization
Server
Issue
Token
Validate
Token
PII
/view/record?id=123
/view/record?id=123
Client
API Gateway
(Resource Server)
Business
Services

OAuth 2.0

1. Client посылает запрос к Authorization Server (AS)
2. Если Client не залогинен (кука нет или не валидный)
    1. AS выдает форму логина и проверяет, что пользователь вошел правильно. Источник пользователей – откуда угодно (DB, LDAP, etc)
3. AS возвращает некоторый Token
4. API Gateway видит запрос с Token
5. API Gateway отправляет Token на AS
6. Если Token верный, AS одобряет.

Доступ к отдельным объектам (Entities) в такой модели регулируется протоколом UMA

OAuth 2.0 + UMA


requesting party
client
authorization server
resource server
resource owner
Set policy conditions (anytime)
Resource request (no access token)
401 response with permission ticket, authz server location
Access token (RPT) request with permission ticket, claim token (push claims)
Claim assessment: FAIL (not authorized)
403 response with new permission ticket, need_info error, redirect_user hint
Redirect user with permission ticket
Follow redirect to authz server
Interactive claims gathering
Redirect back with new permission ticket
Follow redirect to client
RPT request with permission ticket
Claim assessment: OK
Response with RPT and PCT
Response request with RPT
Protected resource
requesting party
client
authorization server
resource server
resource owner

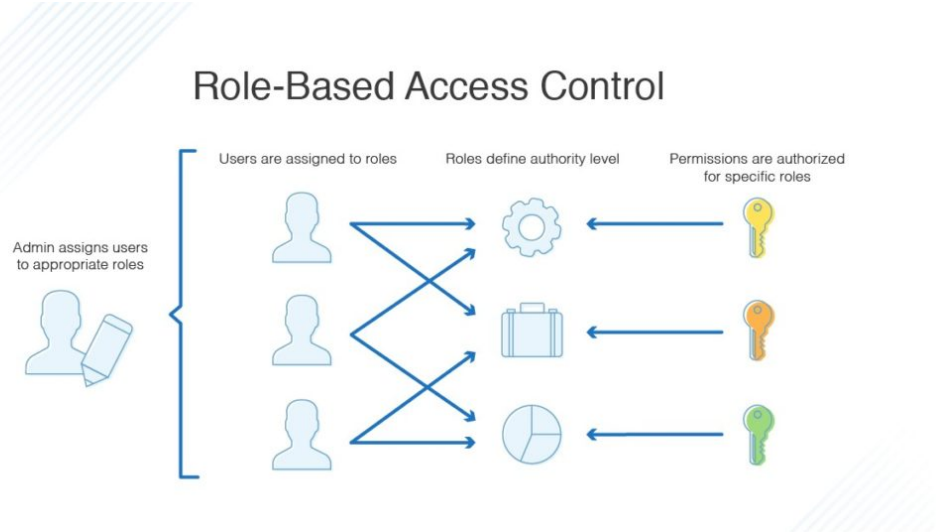
DAC (Discretionary Access Control)

- модель контроля доступа, который может быть передан субъектом другим субъектам. Например, владелец файлов в UNIX может передать права на файлы другим субъектам.

MAC (Mandatory Access Control)

- модель контроля доступа, в которой реализация запрещает субъектам передавать права доступа. Права доступа определяются администратором системы.

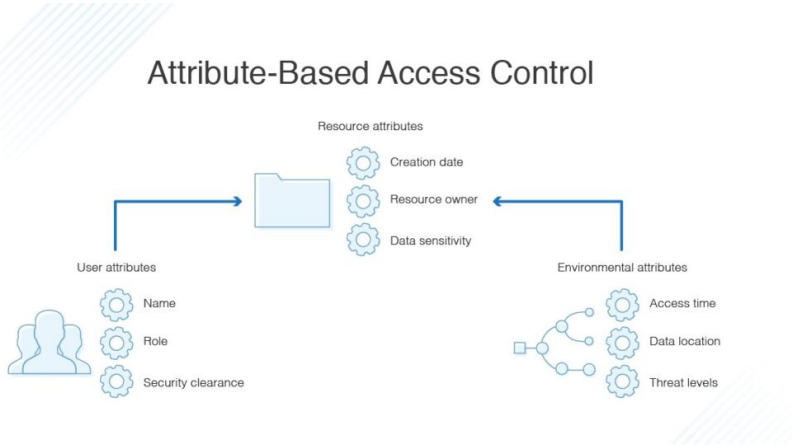
RBAC (Role-Based Access Control)



- Модель контроля доступа, в которой доступ к операции производится на основании того, содержит ли роль субъекта соответствующую привилегию.
- Роль – множество привилегий, доступных пользователям
- Субъекту может быть присвоено несколько (0...n) ролей. Роли может быть присвоено несколько привилегий.

ABAC (Attribute-Based Access Control)

во внимание принимаются свойства (атрибуты) субъекта или объекта, а также свойства окружения (дата, время доступа).



Например:

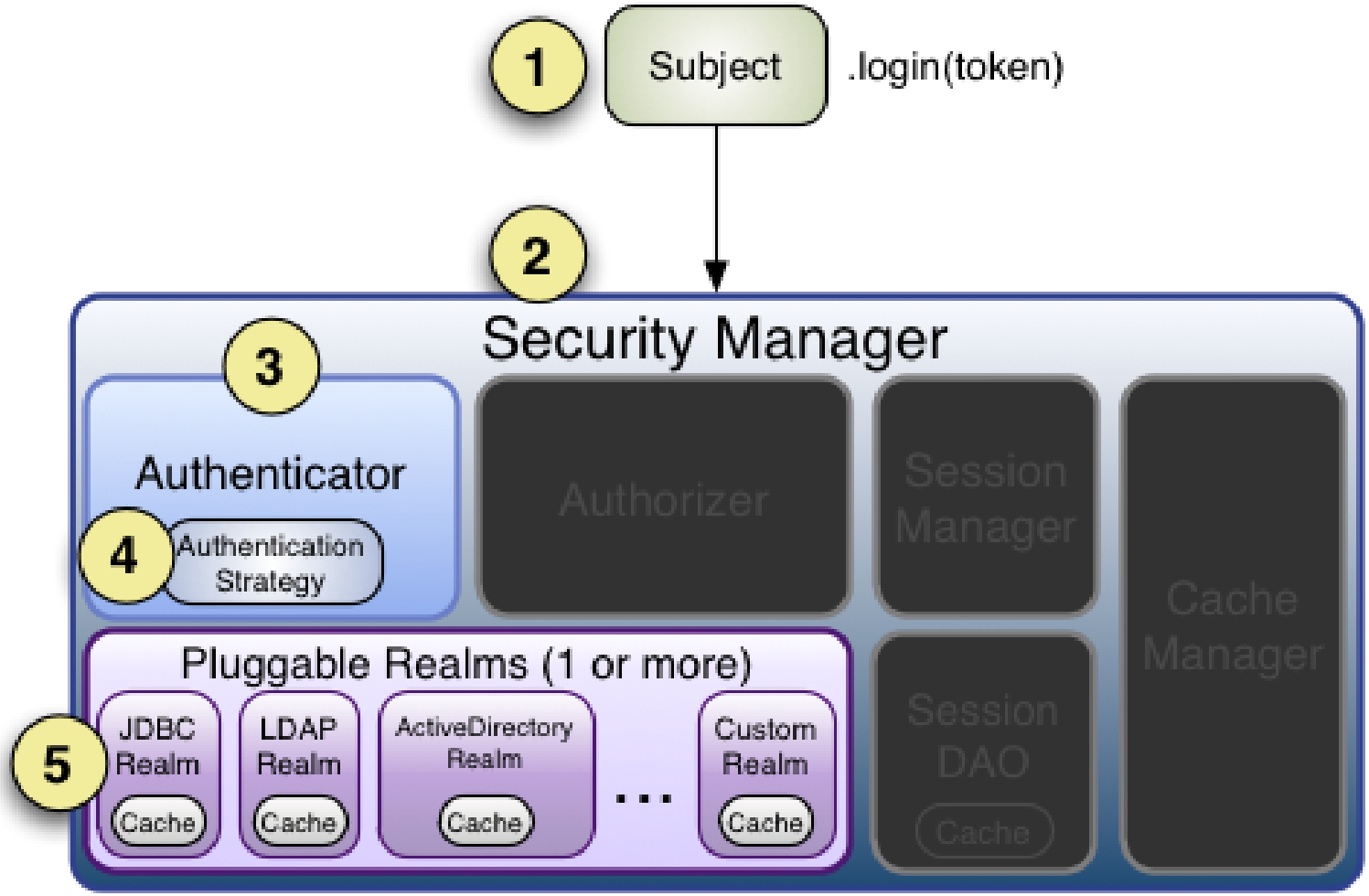
- Пользователь(sub) может просмотреть документ(obj), если он(obj) создан коллегами из того же департамента
- Пользователь(sub) может редактировать документ(obj), если он(subj) создал его(obj) и установлен режим черновика(obj).

Shiro

- Сайт: `https://shiro.apache.org`
- Java, лицензия Apache2
- OAuth2 поддерживается отдельным решением: `https://github.com/bujiio/buji-pac4j`
- Источник данных о пользователях - любой: для этого надо реализовать интерфейс Realm.
- Хорошая документация
- RBAC и item-based permissions (механизм авторизации - любой, прописывается в Java-коде)

Shiro: Механизм аутентификации

Subject – принятый термин для пользователя



Shiro: Механизм аутентификации

1. Приложение вызывает `Subject.login` с аргументом типа `AuthenticationToken` (`token`)
2. экземпляр `Subject` вызывает `SecurityManager.login` с этим аргументом
3. `SecurityManager` вызывает `Authenticator.authenticate(token)`
4. `Authenticator` выбирает `Realm` (источник данных о пользователе) и вызывает его, используя сконфигурированную `AuthenticationStrategy`.
5. Каждый из доступных `Realm`'ов сообщает, поддерживает ли он `token` данного типа, и если да, то для него вызывается `getAuthenticationInfo(token)`, в котором производится попытка аунтетификации.

Shiro: Механизм авторизации

AuthenticationStrategy

Определяет, нужно ли аунтетифицировать по всем Realm или только по одному до первой успешной аунтетификации, и так далее.

Механизм авторизации

Когда вызывается метод `Subject.isPermitted()` или `Subject.checkPermission()`

1. `Subject` делегирует в `SecurityManager`
2. `SecurityManager` делегирует в `Authorizer`
3. `Authorizer` опрашивает `Realm`'ы: a. `Realm` Получает все Permissions и аггрегирует результат, затем делает wildcard matching.

Shiro: минусы

- Отлично подходит для авторизации доступа, но нет возможности программно получить все объекты, к которым пользователь имеет доступ
- В списке рассылки предложили организовать получение всех объектов через свою базу данных (свой интерфейс Realm) – поэтому это решение было отклонено
- Нет поддержки OAuth2 из коробки

Keycloak

- "Коробочное решение" для авторизации с интерфейсом админа, поддерживаемое RedHat
- Правила доступа интегрируются в Spring Boot "из коробки" с несколькими строчками конфига `application.properties`
- Правила доступа задаются в графическом интерфейсе или с помощью JavaScript – с использованием следующих параметров:
  1. Время доступа
  2. Членство в роли (RBAC)
  3. Email пользователя и многие другие атрибуты:

```
var context = $evaluation.getContext();
var identity = context.getIdentity();
var attributes = identity.getAttributes();
var email = attributes.getValue('email').asString(0);
if (identity.hasRealmRole('admin') || email.endsWith('@keycloak.org'))
    $evaluation.grant();
}
```

Keycloak: продолжение

- Права доступа могут задаваться на отдельный ресурс или на "scope" - действие над группой ресурсов (например ="book:read"=), или на группу ресурсов, объединенную общим типом (например ="book"=)
- Есть официальное Java API, будет использовано в Demo-приложении
- Совместим с OAuth2/UMA
- Можно получить список всех доступных пользователю permissions:

```
var ar = new AuthorizationRequest();

    var response = getAuthzClient().authorization().authorize(ar);
    var rpt = getAuthzClient().protection().introspectRequestingPartyTok
    for (Permission granted : rpt.getPermissions()) {
        System.out.println(granted.toString());
    }
```

ORY/Ladon

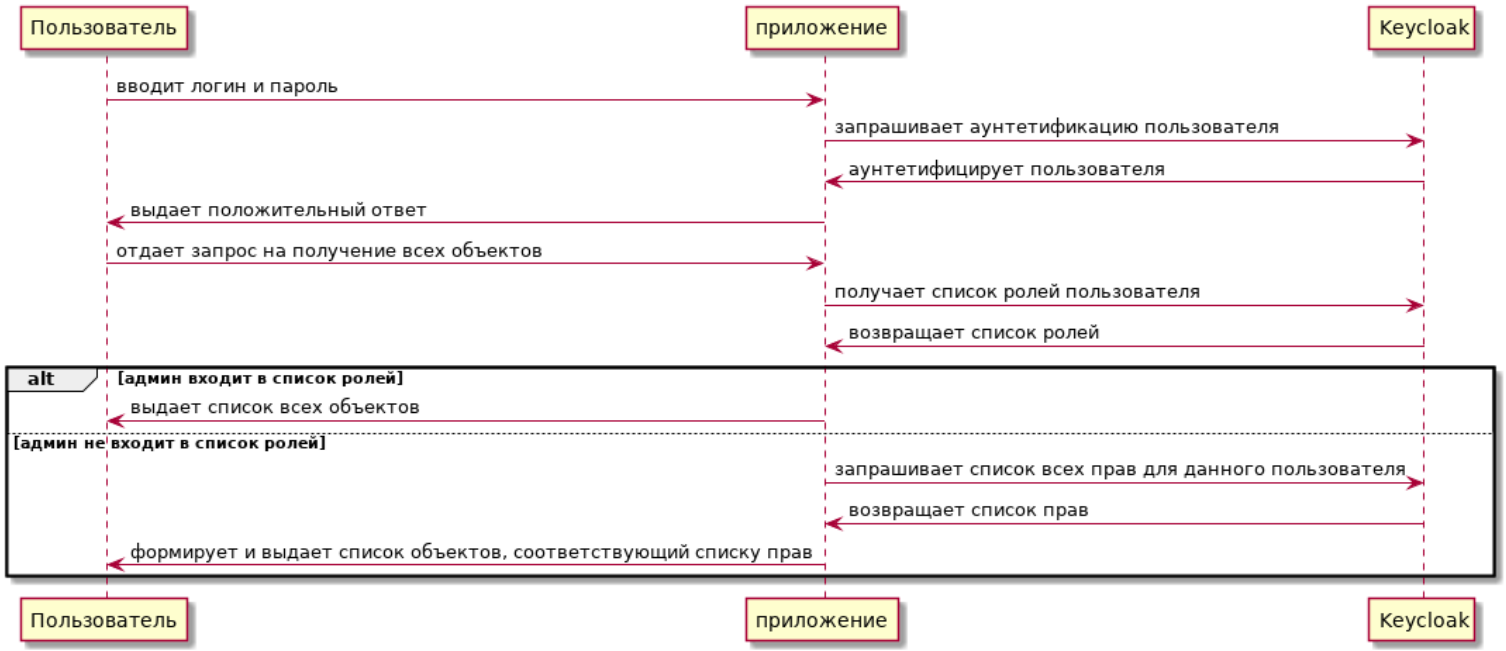
- Golang
- Совместим с OAuth2/UMA
- Источник данных: Любая БД (необходимо реализовать один интерфейс для доступа, есть реализация для Redis)
- `https://github.com/ory/ladon` лицензия Apache2.

Возможно, стоит расмотреть его.

Архитектура

- Приложение написано на Java и имеет JDBC-совместимый источник данных. Информация о правах доступа не хранится в базе данных, но каждой строке ставится в соответствие идентификатор из базы данных Keycloak
- Необходимо придумать, как конвертировать список permissions в ID базы данных, поскольку некоторые permissions могут предоставлять доступ не просто к отдельным ресурсам, но и по widlcard.
- В данном приложении для этого используется сочетание UMA Permissions (DAC) и RBAC (библиотекарь/пользователь)
  - У пользователя производится фильтрация по ID.

Архитектура: взаимодействие



Подводные камни при реализации

- Существует два API Keycloak permissions.
  1. Permissions, редактируемые админом, могут быть простыми DAC/RBAC или сложными JavaScript
  2. User-managed Access – permissions, которые могут быть заданы пользователем для других пользователей над ресурсами, которыми он владеет (это отключаемо, только DAC)

Исходный код приложения

https://github.com/pashazz/library