FACULTY OF ENGINEERING OF THE UNIVERSITY OF PORTO

Handling Security in a Privacy-preserving Health Research Ecosystem

Rostyslav Khoptiy

DISSERTATION



Master in Informatics and Computing Engineering

Supervisor: Ademar Aguiar

Second Supervisor: Artur Rocha

July 27, 2020

Handling Security in a Privacy-preserving Health Research Ecosystem

Rostyslav Khoptiy

Master in Informatics and Computing Engineering

Approved in oral examination by the committee:

Chair: Prof. Jorge Barbosa
External Examiner: Prof. Hélder Gomes
Supervisor: Prof. Ademar Aguiar

July 27, 2020

Abstract

Healthcare data is more available than ever through the power of the internet: a researcher can access the healthcare record of a person, information that could even include his DNA sequence, from his computer. Each country and even each organization might have its repository with this information, with varying levels of access control, if they have any, and without a unified interface to access this information. Accessing healthcare data across various repositories, even if given access for each system, is complicated and time-intensive because of all the different repository access control systems, the authentication schemes they use, and the different legislation regulating data access for each country.

This thesis, which is developed as part of the iReceptor+ project, aims to improve security in DNA healthcare repositories by implementing an access control system with federated authentication on top of a unified HTTP+JSON API interface: the ADC API defined by the AIRR community (mia, ia). There are several repositories in use for DNA based medical data, such as iReceptor Turnkey (ireb, reb), sciReptor (sci, ci) and immuneDB (Lab18, ab18). A repository protected this way allows a researcher, authenticated through an Identity Provider in a Federated Login system, to access different levels of information, such as public data, statistics about DNA information, and DNA sequences. Each level of information can have a different access level associated, generally, the more aggregated the information the easier the access. To each level, access is granted to the researcher by the owner of that information, the resource owner.

Keywords: DNA, medical data, authentication, authorization, access-control, OAuth, OpenID Connect, UMA, HTTP, JSON
ACM Classification: CCS -> Security and privacy -> Security services, CCS -> Security and privacy -> Systems security

Resumo

Os dados de saúde médica estão mais disponíveis do que nunca através do poder da Internet: um investigador pode obter dados médicos de uma pessoa, informações que podem até incluir informações sobre a sua sequência de ADN, através do seu computador. Cada país e até cada organização podem ter os seus próprios repositórios com esses dados, com diferentes estratégias de controlo de acesso, se estes existirem, e sem uma interface unificada para aceder a esses dados. O acesso a esses dados em vários repositórios, mesmo que haja acesso para cada um deles, é complicado e lento, devido a todos os diferentes sistemas de controlo de acesso, os esquemas de autenticação que eles usam e a diferente legislação que regula o acesso a dados médicos de cada país.

Esta tese, sendo inserida no projeto iReceptor+, visa melhorar a segurança de vários repositórios de dados de natureza médica ao implementar um sistema de controlo de acesso com autenticação federada que protege uma interface API HTTP+JSON unificada: a API ADC definida pela comunidade AIRR (mia, ia). Existem vários repositórios usados como o iReceptor Turnkey (ireb, reb), sciReptor (sci, ci) e immuneDB (Lab18, ab18). Um repositório protegido desta forma permite um investigador, autenticado por meio de um *Identity Provider* em um sistema de autenticação federado, aceder a diferentes níveis de informação, como a dados públicos, estatísticas e informações sobre sequências ADN. Cada nível de informação pode ter um nível de acesso diferente associado, geralmente quanto mais agregados forem os dados, mais fácil o acesso. Para cada nível, o acesso é concedido ao investigador pelo proprietário dessa informação: o gerente do recurso.

Palavras Chave: ADN, dados médicos, autenticação, autorização, control de acesso, OAuth, OpenID Connect, UMA, HTTP, JSON
Classificação ACM: CCS -> Security and privacy -> Security services, CCS -> Security and privacy -> Systems security

Acknowledgements

I would like to thank Ademar Aguiar and Artur Rocha for helping me with this dissertation and project and providing me with their support and feedback. I would also like to thank the INESC TEC laboratory for giving me the opportunity to work on the iReceptor Project+ project and to provide my insights into the problem of access control for healthcare-based data. Additionally, I would like to thank the developers and maintainers of the iReceptor Turnkey, sciReptor, and immuneDB projects, specifically Brian Corrie, Scott Christley, and Christian Busse, for giving me feedback on my proposed solutions and validating the requirements and assumptions made for the reference implementation.

Finally I would like to thank my family for supporting me when it was necessary.

Rostyslav Khoptiy

"Truth can only be found in one place: the code." Uncle Bob

Contents

1 Introduction **1**
- 1.1 Context . . . 1
- 1.2 Motivation . . . 2
- 1.3 Objectives . . . 3
- 1.4 Document Structure . . . 3

2 Authentication and Authorization in a Distributed System **5**
- 2.1 Background . . . 5
  - 2.1.1 Authentication . . . 5
  - 2.1.2 Authorization . . . 7
  - 2.1.3 Security Policies . . . 9
- 2.2 Related Technologies . . . 10
  - 2.2.1 OAuth . . . 11
  - 2.2.2 User-Managed Access (UMA) . . . 12
  - 2.2.3 OpenID Connect . . . 16
  - 2.2.4 Security Assertion Markup Language (SAML) . . . 19
  - 2.2.5 JSON Web Tokens (JWT) . . . 20
  - 2.2.6 Keycloak . . . 22
  - 2.2.7 ORCID . . . 22
  - 2.2.8 EGI Check-in . . . 23
- 2.3 Evaluation of technologies . . . 23

3 Problem **25**
- 3.1 Goals . . . 25
- 3.2 Research Strategy . . . 26
- 3.3 Problem Statement . . . 26
- 3.4 Expected Results . . . 26
- 3.5 Validation Strategy . . . 27

4 Healthcare Repositories Multi-layered Security **29**
- 4.1 Requirements . . . 29
  - 4.1.1 Resource Owners . . . 29
  - 4.1.2 Researchers . . . 30
  - 4.1.3 Repositories . . . 30
- 4.2 High-level Architecture . . . 32
- 4.3 Federated Authentication . . . 34
- 4.4 Resource-server Security . . . 35
  - 4.4.1 Repository-level Access Control . . . 38

4.4.2 Middleware Server Access Control 38
4.5 Search Engine Integration 82
4.6 Permissions Dashboard 83

5 Validation **85**
5.1 Requirements Validation 85
5.1.1 Public Information 85
5.1.2 Researcher Login 87
5.1.3 Resource Access Request 88
5.1.4 Access Level Control 93
5.1.5 Multiple Repository Support 96
5.1.6 Additional Requirements 96
5.2 Qualitative Analysis by Experts 97
5.2.1 Access Level Granularity 97
5.2.2 CSV configuration file 98
5.2.3 Synchronization 98
5.2.4 ADC endpoint protection 98
5.2.5 Public information 99
5.2.6 ADC feature support 99
5.2.7 Overall Solution and architecture 100
5.2.8 Additional Insights 100

6 Conclusions **101**
6.1 Key Contributions 102
6.2 Future Work 102

References **105**

A ADC API extension proposal **111**
A.1 Current API 111
A.2 Proposed ADC API extension 112
A.2.1 Why ADC v1.3 is impractical for a middle-ware server 113
A.2.2 New Endpoints 114
A.3 Examples 117

B Fields Access Level Configuration File **129**

C ADC Middleware Server Questionaire **141**

List of Figures

2.1 OAuth 2.0 protocol general flow. . . . 12
2.2 UMA 2.0 protocol general flow. . . . 13
2.3 Example HTTP request to create a resource in the authorization server using the UMA Resource Registration API. On resource creation its UMA resource ID is returned. . . . 15
2.4 Example HTTP request to update a resource in the authorization server using the UMA Resource Registration API. . . . 15
2.5 Example request and response for the DELETE endpoint used to delete resources in the authorization server. No HTTP body is present in the request or response. 15
2.6 Example GET endpoints requests and responses, for reading a resource and listing resource IDs using the UMA Resource Registration API. . . . 17
2.7 Example UMA's token introspection endpoint HTTP request and response. . . . 17
2.8 Example UMA's permission endpoint HTTP request and response. . . . 18
2.9 Example OAuth's token endpoint HTTP request and response. . . . 18
2.10 An example UMA discovery document with only the most important fields from the perspective of a resource server. . . . 18
2.11 OpenID Connect protocol general flow. RP - Relying Party, OP - OpenID Provider. 19
2.12 Example SAML assertion document. . . . 21
2.13 JWT token in plain and encoded form. Here the `iss` and `exp` are registered claims defined by the standard but the `example/value` is a private claim defined by the user. . . . 22
2.14 JWS token in plain and encoded form. The algorithm used for signing is `HMAC SHA-256`, . . . 23

4.1 Proposed solution architecture. Here each node contains both the authorization server, dashboard server and middleware server authorization layer. . . . 34
4.2 Flow for federated login using Keycloak. The flow ends with a login token emitted by Keycloak. Steps 2 and 3 can be omitted if using a local Keycloak account. . . 35
4.3 Flow for accessing a resource from a researcher's perspective. Keycloak acts as both an OIDC IdP and an UMA authorization server. The resource server can either be the repository itself or a middleware server which filters requests for the repository. . . . 36
4.4 Flow for accessing a resource from a researcher's perspective with additional details about the internal workings of the middleware server. . . . 39
4.5 An example request to the endpoint `GET /v1/repertoire/id`. For example: `GET /v1/repertoire/5e53de7f9463684866be6092`, notice the `repertoire_id` field. All of the repertoire's fields are shown. . . . 43

4.6 An example request to the endpoint `GET /v1/rearrangement/id`. For example: `GET /v1/repertoire/5e53dead4d808a03178c7831`, notice the `sequence_id` field. All of the rearrangement's fields are shown. 44
4.7 An example request to the endpoint `POST /v1/repertoire` with a user query. Notice that the repertoires returned don't have all their fields as specified by `"fields"` and that the repertoires returned all match the `"filters"` query. . . 45
4.8 An example request to the endpoint `POST /v1/rearrangement`. Notice that the rearrangements returned don't have all their fields as specified by `"include_fields"` and that the number of repertoires returned is small as specified by `"from"` and `"size"`. 46
4.9 An example facets request to the endpoint `POST /v1/repertoire`. Notice that the `"facets"` parameter interacts with the `"filters"` parameter to restrict the repertoires considered for counting, if it was omitted the counting would be performed over all of the repertoires in the repository. 47
4.10 Example responses from the metadata endpoints. 47
4.11 Researcher's flow for accessing resources using the middleware server and other relevant components. 54
4.12 An example CSV configuration file that can be used by the reference implementation. 55
4.13 An example request to the endpoint `POST /v1/synchronize` with the synchronization password and some example responses. 60
4.14 An example request made by the middleware to the repository on the endpoint `POST /v1/repertoire` to obtain the full list of repertoires. 61
4.15 An example response to the request made by the middleware to Keycloak on the list resource descriptions endpoint. 61
4.16 An example request made by the middleware to Keycloak on the create resource description endpoint. 61
4.17 An example response to the request made by the middleware to Keycloak on the read resource description endpoint. 62
4.18 An example response to the request made by the middleware to Keycloak on the update resource description endpoint. 62
4.19 An example request and response middleware to Keycloak to delete an UMA resource. `rreg` is Keycloak's base path for the resource registration API which can be found in Keycloak's UMA discovery document. 62
4.20 Examples of a filtered repertoire based on the CSV configuration file found in appendix B. 63
4.21 Example user request made to `POST /v1/repertoire` and it's modified version sent to the repertoire in order to emit the permission ticket. Notice how the `"fields"` parameter was modified. 67
4.22 Example user request made to `POST /v1/rearrangement` and it's modified version sent to the repertoire when forwarding the request and returning the resources. 68
4.23 Example user request for repertoire's public fields. The request is sent unmodified to the repository, without needing an RPT token. What are considered public fields depends on the CSV configuration file. 68
4.24 Example user request made to `POST /v1/repertoire` with the facets function. 72

4.25 Example user request made to `POST /v1/repertoire` with the facets function. The added `"in"` filters operator makes sure the resources considered are the ones under the shown study (which is obtained from the UMA ID present in the RPT token). . . . 73
4.26 Example user request made to `POST /v1/repertoire` for repertoire's facets using a public field (`"repertoire_id"`). The request is sent unmodified to the repository, without needing an RPT token. What are considered public fields depends on the CSV configuration file. . . . 73
4.27 Examples response for the `GET /v1/public_fields` public endpoint. . . . 77
4.28 Examples of minimally compliant repertoire and rearrangement endpoint responses either for individual or search endpoints. . . . 78

5.1 Front-end single page SPA overview. . . . 86
5.2 Public front-end endpoints `GET /v1` and `GET /v1/info` response examples. . 86
5.3 Front-end public regular search example request and response for `POST /v1/repertoire`. . . . 87
5.4 Front-end public facets search example request and response for `POST /v1/repertoire`. . . . 87
5.5 Front-end login action example. . . . 88
5.6 Keycloak regular user login action example. . . . 89
5.7 Keycloak third-party IdP login action example. . . . 89
5.8 EGI Check-In login page example. . . . 90
5.9 Front-end logged-in status UI section example. . . . 90
5.10 Front-end researcher resource request example. . . . 90
5.11 Keycloak resource owner's login action example. . . . 91
5.12 Keycloak permissions dashboard resource owner's access grant action example. . 91
5.13 Keycloak permissions dashboard example action for accessing a single study. . . 92
5.14 Keycloak permissions dashboard example action for revoking access. . . . 92
5.15 Keycloak permissions dashboard example action for sharing a resource. . . . 93
5.16 Front-end example repertoire regular search response with filtered repertoires. . . 94
5.17 Front-end example filtered single repertoire response. Notice the lack of the `"sample"` field which was filtered out. . . . 94
5.18 Front-end example filtered repertoire facets response. . . . 94
5.19 Front-end example single rearrangement response. . . . 95
5.20 Front-end example filtered rearrangements regular search. . . . 95
5.21 Front-end example filtered rearrangements facets search. . . . 95
5.22 Front-end example sciReptor filtered repertoire regular search. . . . 96
5.23 Front-end example sciReptor filtered repertoire facets search. . . . 97

List of Tables

2.1 UMA's Resource Registration API. 16

4.1 ADC API overview, current standard (v1). 40
4.2 ADC API to extensions endpoints mapping. 51
4.3 ADC API extensions endpoint descriptions. 52
4.4 Added endpoints specific for the middleware. (A) is password protected which is sent as a Bearer value in the Authorization header and (B) is publicly accessible. . 55

A.1 ADC API overview, current standard (v1.3). 111
A.2 New ADC API proposal overview (v2). 115

Abbreviations

IdP	Federated Identity Provider
USA	United Stated of America
DoD	Department of Defense (USA's)
JSON	JavaScript Object Notation
JWT	JSON Web Token
JWS	JSON Web Signature
ADC	AIRR Data Commons
MiAIRR	Minimal Standard for Adaptive Immune Receptor Repertoire
SSO	Single Sign-On
OIDC	OpenID Connect

Chapter 1

Introduction

Healthcare data is more available than ever before through the power of the internet. By its very nature, healthcare data is very sensitive, and even if de-anonymized can sometimes still be used to identify the person, so special care must be taken when handling this data.

Healthcare data is usually dispersed among various repositories and can be regulated by different legislation.

This thesis describes an approach to handling secure access to DNA healthcare data, in the context of the ADC API and of the studies and patients that provide their DNA information.

1.1 Context

Researchers need to have access to DNA healthcare data, such as the DNA sequence of a person, to do their research but this data is very sensitive. Additionally, special legislation exists, such as the EU Data Protection Directive and the recent GDPR legislation in the EU, that regulates how this data can be accessed and used. Other countries have their own legislation for protecting healthcare data such as the USA's HIPAA.

Healthcare data is mentioned by the GDPR as follows: "Personal data concerning health should include all data pertaining to the health status of a data subject which reveal information relating to the past, current or future physical or mental health status of the data subject. This includes information about the natural person collected in the course of the registration for, or the provision of, health care services as referred to in Directive 2011/24/EU of the European Parliament and of the Council (1) to that natural person; a number, symbol or particular assigned to a natural person to uniquely identify the natural person for health purposes; information derived from the testing or examination of a body part or bodily substance, including from genetic data and biological samples; and any information on, for example, a disease, disability, disease risk,

medical history, clinical treatment or the physiological or biomedical state of the data subject independent of its source, for example from a physician or other health professional, a hospital, a medical device or an in vitro diagnostic test" (Cou16, Cou16).

Researchers might need to access data from multiple repositories and even multiple countries, each potentially having their own regulations.

Researchers can also transform the data or aggregate the data making it harder to identify the original data sources.

This thesis is written as part of Canada's and EU's iReceptor Plus project which has the goal to "... enable researchers around the world to share and analyze huge immunological datasets taken from healthy individuals and sick patients that have been sequenced and stored in databanks in multiple countries" (irea, rea). Among the many partners in the project is the INESC TEC lab in Porto, which has the following objectives for this project, among others:

- Improvement of the HTTP+JSON API interface, ADC API, which is specified by the AIRR Community.
- Improvement of existing DNA repositories with an access control system, including a reference DNA data repository implementing the ADC API.
- Implementation of a mechanism for tracing and logging the researcher's access to and transformations of DNA data.
- Securing the ADC interface with an authentication & authorization system, which is the focus of this thesis.
- Updating of a search engine capable of searching multiple ADC repositories to the new API specification and access control system.

1.2 Motivation

Healthcare data is currently distributed in various repositories making it more difficult for researchers to access the data they need. Each repository might have its own authentication method, interface for accessing DNA information, and different levels of access control and data protection (including possible code vulnerabilities) making it more difficult and time-consuming for a researcher to access data from multiple repositories. Additionally, the owners of DNA data might not have a mechanism for specifying which researcher can have or not access to their medical data, with some data even being publicly accessible.

As with any data system, healthcare data is also sensitive to massive data breaches as happened in the UK's National Health Service (NHS) in the UK where records of 150000 patients were exposed for 1 year (BBC18, BC18).

1.3 Objectives

The main objective of this thesis is to secure a DNA data repository, in the EU context, exposing an ADC API interface with an authentication and authorization system to improve data privacy and security. This objective can be subdivided into:

- Building and evaluating a system for federated authentication for researchers, accepting multiple third-party Identity Providers.
- Protecting ADC endpoints according to the permissions set by the resource owners.
- Exposing a dashboard for resource owners to manage researcher's permissions to each endpoint and resource they own.
- Allowing researchers to use a search engine on the protected repositories.

1.4 Document Structure

This document is structured with the current introductory chapter explaining the context of the project and objectives, followed by the state of the art with the current solutions to the problem of authentication and authorization in chapter 2, the problem description in chapter 3, the proposed solutions in chapter 4, the validation of the chosen solution in chapter 5 and ending with the conclusions and future work in chapter 6.

Chapter 2

Authentication and Authorization in a Distributed System

In this chapter, the current state of the art related to authentication and authorization management, and its related technologies, will be explored.

2.1 Background

There is a need to make a distinction between authentication and authorization which is frequently conflated.

Based on (Shi07, hi07):

- Authentication is defined as "The process of verifying a claim that a system entity or system resource has a certain attribute value", that is, the process of proving that a person is who they say they are.
- Authorization is defined as "An approval that is granted to a system entity to access a system resource", that is, the process of giving access rights to people to access certain resources. This concept is closely related to the concept of access control which is defined as "Protection of system resources against unauthorized access".

2.1.1 Authentication

There are three main authentication categories (authenticators), as defined by (O'G03, 'G03):

- Knowledge-based: based on "What you know". It is based on information that is secret (such as computer passwords) or considered "obscure" (such as a person's favorite color). These have the problem that the information can be guessed, such as when passwords follow a pattern or are re-used. Generally, each time this information is used for authentication it becomes less secret.

- Object-based: based on "What you have". Characterized by physical possession of something, such as physical door keys or computer security tokens. These have the disadvantage that they can be stolen and be used to impersonate the victim. It can be complemented with a password to enhance security.

- ID-based: based on "who you are". Characterized by uniqueness to a person. This includes passports, credit cards, biometric data like fingerprints or eye scans, etc. These have the advantage of being difficult to copy or forge but are difficult to replace if lost.

Multi-factor authentication is defined as a combination of these authenticators used to enhance security.

For remote, server-based authentication passwords (Knowledge-based) are currently the main form of authentication. Even though they have some drawbacks, such as sometimes being easy to guess or being reused, they are still the most cost-effective, convenient, and practical method of authentication compared to the alternatives (BHVS15, HVS15), at least as a first step in the authentication flow. The first known use of a password for authentication in a computer system was recorded in 1961 to secure access to a time-shared mainframe at MIT (BHVS15, HVS15).

Currently, the main form of this authentication is done using a username and password combination. A username can be any string chosen by the person to identify himself, on the internet, it is frequently the email of the person. The password is used as a challenge to authenticate the person with the idea that only the actual person would know the password.

Passwords as used in this manner are usually hashed and salted when stored in the machine. Salting refers to generating a random string and appending it to the password to avoid dictionary attacks. A well-chosen cryptographically secure hashing function (such as BCrypt) should make it impossible in practice, that is, it should take several lifetimes of brute-forcing the hash with the most powerful computer, to guess the original salted password from the hash (PM99, M99). The reason to store password hashed instead of in plain-text is to not expose the original password if there is a breach of the password hashes, such as in a web application's database breach (SK15, K15).

If a password is well-chosen, that is, it is a long enough random string to mitigate password guessing (the *random user model*), and there are no side-channel attacks (such as phishing, theft, eavesdropping, etc), the only viable method of getting the password is by brute-forcing the authentication system (*online* attack) or by brute-forcing the hash function (*offline* attack). If the attacker has access to the password hashes (*offline*), the attack is more powerful because the attacker is only limited by his computational resources whereas in a *online* attack the attacker needs to go through the authentication system for each guess. If the password is well chosen, it should be computationally infeasible to obtain the original password using brute-forcing (BHVS15, HVS15).

Computer tokens (Object-based) are also frequently used in web-applications and are frequently obtained after a user has logged in with a username and password. These can be used to access protected information and have advantages over using the user credentials for each request, such as the ability to individually revoke tokens or emit tokens that only allow restricted access

(compared to the user's credentials). Additionally, as mentioned before, passwords in general loose some of their secrecy with each use, so exchanging it for a token can remedy this somewhat. As defined by (KHJK16, HJK16) tokens can be classified as perishable, session, access, or refresh tokens, with different purposes, lifetimes, reusability, renewability, etc.

Federated login

Federated login, also referred to as single sign-on (SSO), is the process of a system outsourcing the process of authenticating a person to another system that it trusts. On a website, this is frequently done by the original website redirecting the person to a new website where he/she will log in and then redirect back to the original system with the logged-in information (Shi07, hi07).

Single sign-on systems have several advantages over systems that implement their own login system (ACC+08, CC+08):

- No need for the server to store the username and password of the users, which is frequently leaked.
- Delegation of the registration and login mechanisms, which can be exploited if incorrectly implemented, to a server that specializes in this process.
- Improved security from the end-user not reusing his username-password combination on the consuming server, which frequently happens.
- Improved user experience from the user not having to create an account and remember the credentials for each new service.

2.1.2 Authorization

Authorization refers to the process of granting access to a resource to an entity. As authentication, it has a long history before the age of computers, most prominently in the military, and the government where guarding classified information is imperative. It was in the USA's DoD that a lot of access control systems were developed.

As mentioned before, this concept is related to access control.

It is useful to define some concepts related to authorization which will be used throughout this thesis, as defined by OAuth (Har12, ar12):

- Resource Owner: "An entity capable of granting access to a protected resource. When the resource owner is a person, it is referred to as an end-user."
- Resource Server: "The server hosting the protected resources, capable of accepting and responding to protected resource requests using access tokens."
- Authorization Server: "The server issuing access tokens to the client after successfully authenticating the resource owner and obtaining authorization."

- Client: "An application making protected resource requests on behalf of the resource owner and with its authorization. The term "client" does not imply any particular implementation characteristics (e.g., whether the application executes on a server, a desktop, or other devices)." Throughout this thesis, this concept is used more loosely to refer to any application acting on behalf of the person accessing resources from the resource server.

Access Control Systems

An access control system is a system for selective restriction of access to resources. The systems described here are not necessarily mutually exclusive, for example, an RBAC (Role-based Access Control) system can implement a DAC (Discretionary Access Control) and a MAC (Mandatory Access Control) system.

Access-control list (ACL)

For each resource in the system, there is an associated list describing all the users that are allowed to manipulate the system and the permissions of each user relating to the resource (Shi07, hi07).

In the example of a file system a file might contain a list like (Alice: read,write; Bob: read) where the file can be read or written by Alice but only read by Bob.

This model has the disadvantage of being relatively complex, compared to the others, because each resource in the system needs to be individually configured for access control, which can be a lot of work if there is a lot of resources in the system.

Discretionary Access Control (DAC)

DAC is a type of access control defined by USA's Trusted Computer System Evaluation Criteria as "a means of restricting access to objects based on the identity of subjects and/or groups to which they belong. The controls are discretionary in the sense that a subject with a certain access permission is capable of passing that permission (perhaps indirectly) on to any other subject (unless restrained by mandatory access control)" (oDS85, DS85), that is, resources in a system belong to a user or a group and a user can give to another user the ownership of a resource he owns.

This access control system was born in the USA's Department of Defense.

It can also be described using a formal language as introduced by (FK92, K92).

Mandatory Access Control (MAC)

MAC is defined by USA's Trusted Computer System Evaluation Criteria as "a means of restricting access to objects based on the sensitivity (as represented by a label) of the information contained in the objects and the formal authorization (i.e., clearance) of subjects to access information of such sensitivity" (oDS85, DS85). An access policy is set by an administrator and users of the system cannot override this policy to share their access with another user (in contrast to DAC).

Role-based Access Control (RBAC)

The main idea behind this access control system is the concept of user roles. User roles can be defined as "a set of actions and responsibilities associated with a particular working activity. Then, instead of specifying all the accesses each individual user is allowed, access authorizations on objects are specified for roles" (AS00, S00). In this system, each role specifies the operations that can be performed on each resource.

A role can have the same permissions as another role, so many systems implement an inheritance mechanism creating a role hierarchy.

This system has the advantage of easier access control administration given that roles and their privileges are relatively static within an organization and they follow more closely how access control is performed in the real world (FK92, K92). In this system, an administrator needs to simply add or remove roles from a user to change his access permissions.

2.1.3 Security Policies

As defined by (Shi07, hi07) a security policy is "a set of policy rules (or principles) that direct how a system (or an organization) provides security services to protect sensitive and critical system resources", in other words, a specification of the requirements for considering a system or organization secure. Security policies are closely related to access control policies, a policy can, for example, implement strategies for RBAC or other access controls.

The models described here have applications both in computer systems and in the real world in organizations.

2.1.3.1 Bell-LaPadula model

The Bell-LaPadula model focuses on ensuring system confidentiality, developed originally for the USA's military. According to this model, the entities in an information system are divided into subjects and objects, additionally both objects and subjects can be labeled with security levels and categories (or compartments). The Security levels are an ordered list from lowest to highest confidentiality, each subject or object has one and only one security level. Categories, which represent the scope, are unordered and each subject or object can have a set of 0 or more categories. Additionally, the concept of secure and insecure system states is defined and proven that system security is preserved with system transitions from secure to secure states (BLP76, LP76).

Three rules are defined to describe this security policy:

1. Simple security condition: a subject can only read objects in a security level equal or inferior to its own level and of equal or narrower scope (category set of an object is the subset of category set of the subject).

2. Star condition: a subject can only write to objects in a security level equal or superior to its own level and of equal or broader scope (category set of an object is the superset of category set of the subject).

3. Discretionary security condition: a subject may access objects for which it has the corresponding individual access permission.

Rules 1 and 2 can be described as MAC rules and rule 3 as DAC.

2.1.3.2 Biba model

The Biba model focuses on ensuring system integrity. It is similar to the Bell-LaPadula model in that there are subjects and objects with integrity (security) levels and categories. Integrity levels represent the level of trust in the integrity of the object or actions of the subject. (Bib77, ib77).

The rules, of which the first two are a dual of the Bell-LaPadula model:

- Simple Integrity Property: a subject may read objects with an equal or superior integrity level.
- Star Integrity Property: a subject may write on objects of an equal or inferior level.
- Invocation Property: a subject at one integrity level is prohibited from invoking or calling up a subject at a higher level of integrity.

2.1.3.3 Chinese Wall model

The Chinese Wall model focuses on confidentiality and it was developed to describe the security policies present in the commercial world that could not be expressed correctly using the Bell-LaPadula model. In addition to the concept of subjects and objects, the model defines, using commercial terminology, the concept of company dataset, which is the set of all objects that concern the same corporation, and the concept of conflict of interest class, which is the set of all companies or company datasets that compete with each other (BN89, N89).

The model is then defined by the rules:

- Simple Security: a subject can read objects from at most one company dataset in each conflict of interest class. That is, if there are 2 companies in a conflict of interest class and 2 companies in another conflict of interest class, the subject can only read from 1 company in the first and 1 company in the second conflict of interest list.
- Sanitized Information: a subject can read objects from multiple company datasets in the same conflict of interest class if the information is sanitized.
- Star Property: a subject can write to an object if, using the simple security rule, he can read the object and he cannot read objects with unsanitized information from any other company other than to the one which he wants to write.

2.2 Related Technologies

There are various standards for performing authentication and authorization and various libraries and tools which implement these standards.

2.2.1 OAuth

OAuth is an open standard for federated authorization, as described by (Har12, ar12): "The OAuth 2.0 authorization framework enables a third-party application to obtain limited access to an HTTP service, either on behalf of a resource owner by orchestrating an approval interaction between the resource owner and the HTTP service, or by allowing the third-party application to obtain access on its own behalf". The current version in use, version 2, is designed for use over HTTP.

This standard attempts to remedy the lack of security present in the basic model, where a third party (client) that needs to access a resource in a resource server from the resource owner would use the credentials (username and password) of the owner. This has several issues, which are common with schemes using plain passwords, such as:

- The client needs to store the credentials, typically in plain-text.
- Resource servers are required to support password-based authentication.
- The client gains access to protected resources which it does not need to access.
- Resource owners cannot revoke a single third party without revoking all third parties.
- The compromise of any third party would result in the compromise of the resource owner's credentials.

OAuth attempts to solve this by introducing an authorization layer in which the third party accessing a resource is issued credentials specific to the resources it needs to access, which are different from the resource owner's credentials and which only allow access to resources specified by the resource owner.

The third-party (client) is issued an access token string with a specific scope and lifetime, as specified by the resource owner. The client uses this token to access protected resources. This token is issued by an authorization server, which is trusted by both the client and the resource owner.

To access a protected resource a client, in the general case, must perform 3 steps (as can be seen in Fig. 2.1):

1. Request an authorization grant from the resource owner, which the owner can grant or deny. This step can be done with the aid of an authorization server.
2. Request an access token from the authorization server using the authorization grant from the previous step. The server will validate the authorization grant and decide whether to grant the access token.
3. Request the protected resource from the resource server using the access token from the previous step. The server can reject the request if the token is invalid.

Before making use of the flow the client needs to register itself with the authorization server to obtain the authentication credentials it can use to authenticate itself with this server.

OAuth 2 has various extensions defined, such as:

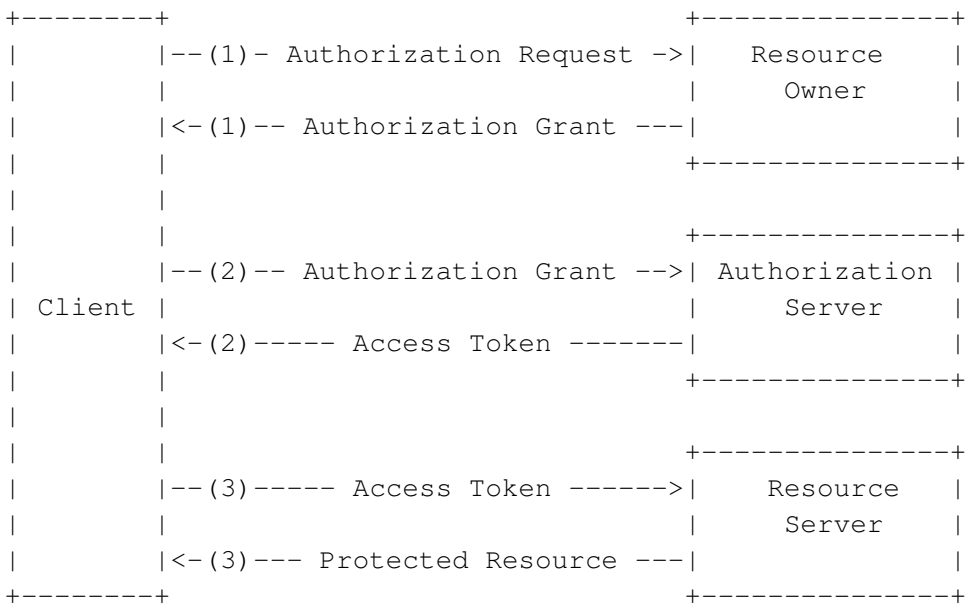


Figure 2.1: OAuth 2.0 protocol general flow.

- Introspection Endpoint: defines an endpoint on the Authorization server which allows a client to query metadata about a token (such as access or refresh token). This metadata includes whether the token is active (or valid) and may include data such as the scope list, the issuing and expiration time among others (JR15, R15).

- Authorization Server Metadata: defines an endpoint that returns a JSON document with information about the server such as the paths of the authorization and token endpoints, the scopes supported among other data (JSB18, SB18).

2.2.2 User-Managed Access (UMA)

User-Managed Access (UMA) 2.0 is an open standard for federated authorization based on OAuth, with the goal of enhancing OAuth to allow party-to-party authorization: the resource owner and the requesting party (user of the client) might be different people and the goal of allowing asynchronous interaction: when the client requests an authorization token the resource owner doesn't have to be present to grant the access (MR17b, R17b).

The UMA standard defines an extension to OAuth 2 for client interaction with the authorization server (MR17b, R17b) and defines a flow for resource server interaction with the authorization server (MR17a, R17a).

To access a protected resource, a requesting party can perform these steps, as can be seen in Fig. 2.2:

1. Access the protected resource in the resource server using either no token or an invalid or expired RPT token. The requesting party token (RPT) is an access token associated with the

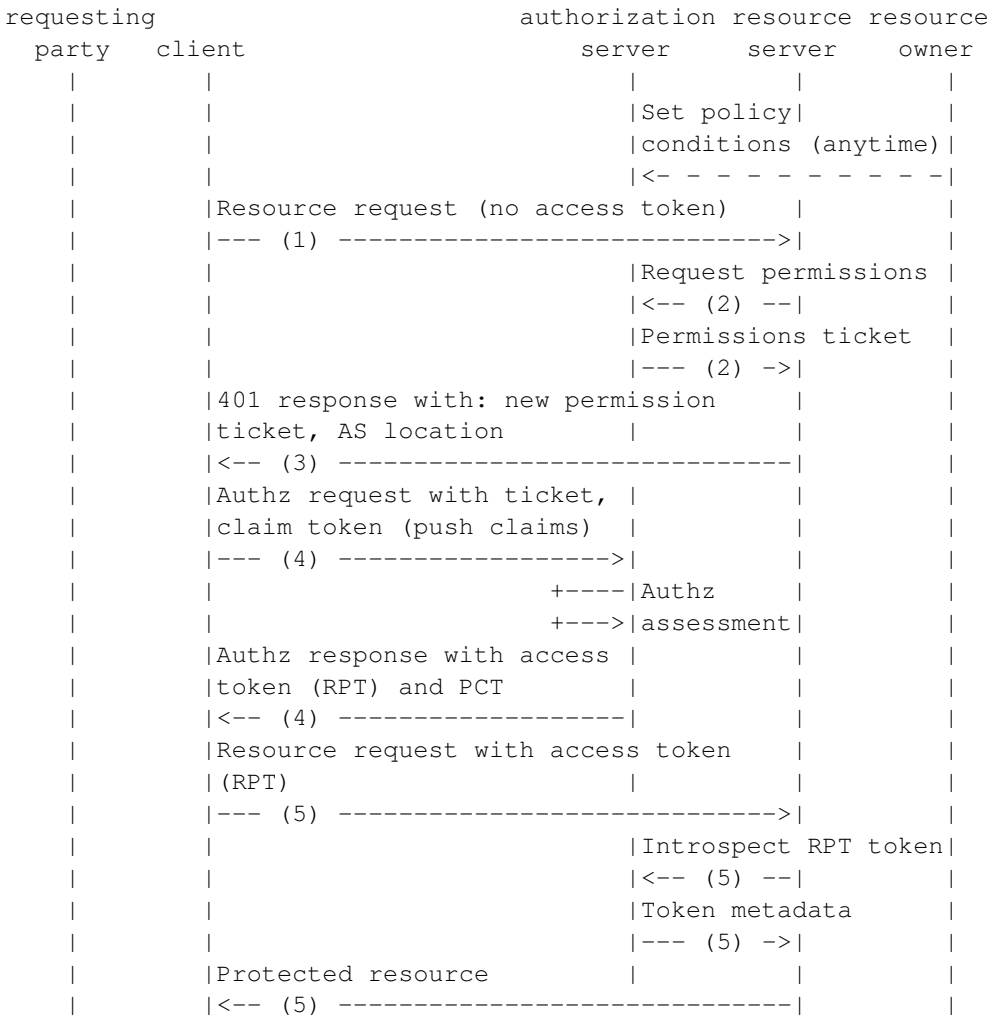


Figure 2.2: UMA 2.0 protocol general flow.

UMA grant. An RPT is unique to a requesting party, client, authorization server, resource server, and resource owner. An RPT token might be associated with multiple resources.

2. The resource server, on receiving incorrect authorization contacts the authorization server to request a ticket token. In this request, the resource server must specify the associated resource (which must be previously registered) and the scopes associated with the resource to be included in the ticket. Multiple resources along with their scopes might be requested.

3. The resource server returns the ticket along with the location of the authorization server for the client to perform an authorization check.

4. The client requests an RPT token from the authorization server, by sending the ticket token from the previous step, a list of requested scopes, and any additional claims such as an OpenID Connect ID token. The process of user authentication, or proving of certain claims about the requesting party, can be done in any way the authorization server specifies, including an interactive login or captcha or claims such as OpenID Connect tokens or SAML

claims. The authorization server checks the scopes that are available for the resource associated with the ticket, along with the scopes requested by the resource server and client and either returns an access token (RPT token) or an error. The scopes associated with the RPT token might be a subset of the requested scopes depending on the policy set for the user by the resource owner. Scopes requested but not granted to the client can be placed as a request by the user on the system, for future resource owner validation. For each resource in the authorization server, there are rules for which users can access and which scopes they have access to, with the details of associating users with scopes being outside the scope of this specification. Additionally, a PCT (persisted claims token) token is returned to simplify future authorization requests, and optionally a refresh token.

5. The client using the RPT token accesses the same resource from step 1. The resource server then validates the RPT token by contacting the authorization server and obtains the scopes granted by the authorization for that RTP token. The server then decides whether to grant access based on its own internal rules, details which are outside the scope of this specification.

A UMA enabled authorization server must publish metadata, called the UMA discovery document from now on, as specified by (JSB18, SB18) which must include the permissions and resource registration URI and optionally the introspection endpoint if supported by the server, among other data. This document is accessible by clients and resource servers. See the example in figure 2.10, which was obtained from a Keycloak instance. The path for this endpoint must be specified somewhere by the authorization server.

The resource server, before any client requests, must register with the authorization server the resources it wants to protect. There are endpoints for creating, viewing, listing, updating and deleting resources, see table 2.1 for the full list of resource registration endpoints and figures 2.3, 2.5, 2.6 and 2.4 for examples.

As shown in table 2.1, using a PAT a user can list, get, create, update, and delete resources manged by the authorization server where `rreguri` is the base URL under which this API is hosted and it can be obtained from the UMA discovery document.

Each resource has associated an ID and a list of scopes specific to that resource, which can be any string or URI, among other data. When the scope is a URI (to any server) it must point to a JSON document that has information such as a description, icon image, and name for the scope, which can be useful for standardization. When creating a resource the response contains the resource ID which must be used in subsequent requests to manage the resource or request a ticket for the resource. This is done using the base OAuth 2 protocol where the client is the resource server and the resource owner grants authorization to the resource server to create and manage resources and use the endpoints associated with ticket creation and token introspection (mainly for the RPT token).

The token introspection endpoint as defined in (JR15, R15) is extended so that when the `active` value is set to `true` the `scope` value is replaced with the `permissions` value containing

```
// POST /rreg/
// Authorization: Bearer MHg3OUZEQkZBMjcx
{
   "resource_scopes":[
      "read-public",
      "post-updates",
      "read-private",
      "http://www.example.com/scopes/all"
   ],
   "icon_uri":"http://www.example.com/icons/
       ↪ sharesocial.png",
   "name":"Tweedl Social Service",
   "type":"http://www.example.com/rsrcs/
       ↪ socialstream/140-compatible"
}
```

(a) Example Request

```
// 201 Created
{
   "_id":"KX3A-39WE",
   "user_access_policy_uri":"http://as.
       ↪ example.com/rs/222/resource/KX3A
       ↪ -39WE/policy"
}
```

(b) Example Response

Figure 2.3: Example HTTP request to create a resource in the authorization server using the UMA Resource Registration API. On resource creation its UMA resource ID is returned.

```
// PUT /rreg/9UQU-DUWW HTTP/1.1
// Authorization: Bearer 204c69636b6c69
{
   "resource_scopes":[
      "http://photoz.example.com/dev/scopes/
          ↪ view",
      "public-read"
   ],
   "description":"Collection of digital
       ↪ photographs",
   "icon_uri":"http://www.example.com/icons/
       ↪ sky.png",
   "name":"Photo Album",
   "type":"http://www.example.com/rsrcs/
       ↪ photoalbum"
}
```

(a) Example Request

```
// 200 OK
{
   "_id":"9UQU-DUWW"
}
```

(b) Example Response

Figure 2.4: Example HTTP request to update a resource in the authorization server using the UMA Resource Registration API.

```
// Request
DELETE /rreg/9UQU-DUWW
Authorization: Bearer 204c69636b6c69

// Response
204 No content
```

Figure 2.5: Example request and response for the DELETE endpoint used to delete resources in the authorization server. No HTTP body is present in the request or response.

Endpoint	Type
POST rreguri/	Create resource description
GET rreguri/_id	Read resource description
PUT rreguri/_id	Update resource description
DELETE rreguri/_id	Delete resource description
GET rreguri/	List resource descriptions

Table 2.1: UMA's Resource Registration API.

a list of resources and their individual IDs and scopes, among other values. See the example in figure 2.7 where the request's token is the RPT token that the resource server wants to introspect. In the figure the response's `"active"` indicates whether the token is still active, when the value is `false` the other fields may be omitted. In the figure the request's `/introspect` path is the URL under which this endpoint is hosted by the authorization server, it can be obtained from the UMA discovery document.

The permission endpoint, as used in step 2, is used by the resource server to issue a request to the authorization server to validate the user's access to the specified resources and scopes. The resource server can specify any amount of resources and any scopes that the resources have registered. The ticket returned by this endpoint to the resource server is intended to be returned to the user to be used by him, in conjunction with his login token to obtain an RPT token. As can be seen in the example in figure 2.8. In the figure the request's `/perm` path is the URL under which this endpoint is hosted by the authorization server, it can be obtained from the UMA discovery document.

The OAuth access token used in these endpoints by the resource server is called the PAT (protection API access token). It can be obtained using OAuth's token endpoint, as can be seen in the example in figure 2.9. This endpoint is used to authenticate the client and obtain an access token, the PAT, for further requests. There are other authentication formats than the one in the figure. In the figure, the request's basic authorization token contains the client's client ID and client secret, which is obtained by previously registering the client with the authorization server.

The resource owner is responsible for setting the policies for user access on the authorization server, which should be done ideally before any client requests.

2.2.3 OpenID Connect

OpenID Connect, also abbreviated as OIDC in this thesis, is an open standard for federated authentication implemented as an extension of OAuth. As described in (SNB+14, NB+14): "OpenID Connect 1.0 is a simple identity layer on top of the OAuth 2.0 protocol. It enables Clients to verify the identity of the End-User based on the authentication performed by an Authorization Server, as well as to obtain basic profile information about the End-User in an interoperable and REST-like manner."

```
// Request
// GET /rreg/KX3A-39WE
// Authorization: Bearer 204c69636b6c69

// Response
// 200 OK
{
   "_id":"KX3A-39WE",
   "resource_scopes":[
      "read-public",
      "post-updates",
      "read-private",
      "http://www.example.com/scopes/all"
   ],
   "icon_uri":"http://www.example.com/icons/
       ↪ sharesocial.png",
   "name":"Tweedl Social Service",
   "type":"http://www.example.com/rsrcs/
       ↪ socialstream/140-compatible"
}
```

(a) Example Request and Response for reading a single resource

```
// Request
// GET /rreg/
// Authorization: Bearer 204c69636b6c69

// Response
// 200 OK
[
   "KX3A-39WE",
   "9UQU-DUWW"
]
```

(b) Example Request and Response for listing all the resources in the authorization server.

Figure 2.6: Example GET endpoints requests and responses, for reading a resource and listing resource IDs using the UMA Resource Registration API.

```
// POST /introspect
// Authorization: Bearer 204c69636b6c69

token=sbjsbhSSJHBSUSSJHVhjsgvhsgvshgsv
```

(a) Example Request

```
// 200 OK
{
   "active":true,
   "exp":1256953732,
   "iat":1256912345,
   "permissions":[
      {
         "resource_id":"112210f47de98100",
         "resource_scopes":[
            "view",
            "http://photoz.example.com/dev/
                ↪ actions/print"
         ],
         "exp":1256953732
      }
   ]
}
```

(b) Example Response

Figure 2.7: Example UMA's token introspection endpoint HTTP request and response.

```
// POST /perm
// Authorization: Bearer 204c69636b6c69
[
   {
      "resource_id":"7b727369647d",
      "resource_scopes":[
         "view",
         "crop",
         "lightbox"
      ]
   },
   {
      "resource_id":"7b72736964327d",
      "resource_scopes":[
         "view",
         "layout",
         "print"
      ]
   }
]
```

(a) Example Request

```
//  201 Created
{
   "ticket":"016f84e8-f9b9-11e0-bd6f-0021
       ↪ cc6004de"
}
```

(b) Example Response

Figure 2.8: Example UMA's permission endpoint HTTP request and response.

```
// POST /token

// Authorization: Basic
    ↪ czZCaGRSa3F0MzpnWDFmQmF0M2JW
// Content-Type: application/x-www-form-
    ↪ urlencoded

grant_type=client_credentials
```

(a) Example Request.

```
// 200 OK
{
    "access_token":"2YotnFZFEjr1zCsicMWpAA",
    "token_type":"example",
    "expires_in":3600,
    "example_parameter":"example_value"
}
```

(b) Example Response

Figure 2.9: Example OAuth's token endpoint HTTP request and response.

```
// GET /auth/realms/master/.well-known/uma2-configuration
{
  "issuer": "http://localhost:8082/auth/realms/master",
  "token_introspection_endpoint": "http://localhost:8082/auth/realms/master/protocol/
      ↪ openid-connect/token/introspect",
  "resource_registration_endpoint": "http://localhost:8082/auth/realms/master/authz/
      ↪ protection/resource_set",
  "permission_endpoint": "http://localhost:8082/auth/realms/master/authz/protection/
      ↪ permission",
  "introspection_endpoint": "http://localhost:8082/auth/realms/master/protocol/openid-
      ↪ connect/token/introspect"
}
```

Figure 2.10: An example UMA discovery document with only the most important fields from the perspective of a resource server.

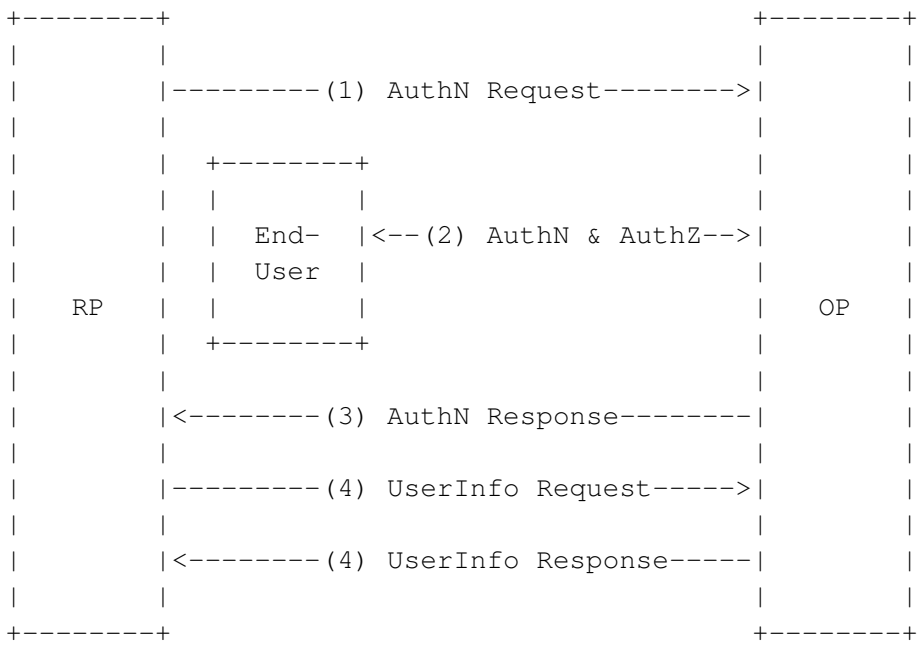


Figure 2.11: OpenID Connect protocol general flow. RP - Relying Party, OP - OpenID Provider.

The goal of this standard is to allow a Client (Relying Party - RP) to be able to authenticate an End-User using an authorization server from OAuth (the OpenID Provider - OP). The client is issued an ID Token from the authorization server on success, which is a JWS (signed JWT) token that can itself contain various Claims about the End-User (such as the name, email, etc). Further Claims about the End-User can be obtained by the Client using an access token if provided by the authorization server.

In general, the OpenID Connect protocol follows the steps below, as can be seen in Fig. 2.11:

1. The Client sends a request to the OP.
2. The OP authenticates the End-User and obtains authorization from him.
3. The OP responds with the ID token and optionally an Access Token.
4. The Client can optionally request additional Claims about the End-User from the OP using the provided Access Token.

2.2.4 Security Assertion Markup Language (SAML)

SAML (Security Assertion Markup Language), currently on version 2 is a widely used and open standard for authentication, authorization, and exchanging information about a subject. Information is exchanged using SAML assertions (statements), defined in XML documents with a schema, containing information, usually about a subject (user), between an Identity Provider (SAML authority) and the service provider (relying party) (CHL+05a, HL+05a).

As can be seen in the example in figure 2.12, there are some important tags in a SAML assertion document such as the `saml:Subject` tag which identifies the user, the `ds:Signature` tag which contains the digital signature for the document and the `saml:AuthnStatement` tag which asserts that the subject was authenticated at the identity provider.

In addition to the assertions format, the standard defines protocol messages, in XML, to be used in a variety of transport protocols. These can be used for querying for specific assertions from the SAML authority, for performing authentication, performing a session logout, etc.

The standard defines various bindings for the SAML assertions and protocol messages to be used in various protocols such as SOAP, HTTP POST, among others, that specify how SAML messages are to be embedded in these protocols (CHL+05b, HL+05b).

The standard specifies a metadata message format, in XML, that can include information about the location of HTTP endpoints for authentication, if the identity provider is self-certified and the digital certificate location, among others (CHL+05c, HL+05c).

SAML responses, as part of the SAML protocol messages, can be unsigned or signed with public keys, and SAML assertions can be unsigned, signed, and even encrypted.

2.2.5 JSON Web Tokens (JWT)

JSON Web Tokens (JWTs) are described by (JMB+15b, MB+15b) as: "a compact, URL-safe means of representing claims to be transferred between two parties. The claims in a JWT are encoded as a JSON object ...". JWTs are simply JSON objects (see Fig. 2.13 for an example) with a defined schema and some object fields with defined semantics (claims) encoded as a base64 string (see (Jos06, os06) for more information about base64). The JSON object keys can be as defined in the standard (registered claims) or as defined by the users and registered in the IANA to avoid name collision (public claims) or without registration (private claims) where the producers and consumers need to agree on their semantics. None of the registered claims are mandatory but are useful for interoperability.

2.2.5.1 JSON Web Signature (JWS)

JSON Web Signature (JWS) (JMB+15a, MB+15a) are JWT tokens that are signed using a signed hash. Both public and symmetric keys are supported. JWS tokens follow the format `header.payload.signature` where the header, payload, and signature are base64 encoded and separated by a `'.'` character. The header is a JSON object containing claims about the type of the token (JWT) and signing algorithm used, the payload contains the JWT token and the signature contains the signed hash of the header and payload.

These types of tokens are used to prove that the token was generated by the original entity without being modified (integrity). They expose the claims to the holders of the tokens.

The standard defines algorithms for signing while also allowing for new algorithms to be adopted. Some of the algorithms in the standard are the `HMAC with SHA-2` for symmetric

```
<saml:Assertion
   xmlns:saml="urn:oasis:names:tc:SAML:2.0:assertion"
   xmlns:xs="http://www.w3.org/2001/XMLSchema"
   ID="_d71a3a8e9fcc45c9e9d248ef7049393fc8f04e5f75"
   Version="2.0"
   IssueInstant="2004-12-05T09:22:05Z">
   <saml:Issuer>https://idp.example.org/SAML2</saml:Issuer>
   <ds:Signature
     xmlns:ds="http://www.w3.org/2000/09/xmldsig#">...</ds:Signature>
   <saml:Subject>
     <saml:NameID
       Format="urn:oasis:names:tc:SAML:2.0:nameid-format:transient">
       3f7b3dcf-1674-4ecd-92c8-1544f346baf8
     </saml:NameID>
     <saml:SubjectConfirmation
       Method="urn:oasis:names:tc:SAML:2.0:cm:bearer">
       <saml:SubjectConfirmationData
         InResponseTo="aaf23196-1773-2113-474a-fe114412ab72"
         Recipient="https://sp.example.com/SAML2/SSO/POST"
         NotOnOrAfter="2004-12-05T09:27:05Z"/>
     </saml:SubjectConfirmation>
   </saml:Subject>
   <saml:Conditions
     NotBefore="2004-12-05T09:17:05Z"
     NotOnOrAfter="2004-12-05T09:27:05Z">
     <saml:AudienceRestriction>
       <saml:Audience>https://sp.example.com/SAML2</saml:Audience>
     </saml:AudienceRestriction>
   </saml:Conditions>
   <saml:AuthnStatement
     AuthnInstant="2004-12-05T09:22:00Z"
     SessionIndex="b07b804c-7c29-ea16-7300-4f3d6f7928ac">
     <saml:AuthnContext>
       <saml:AuthnContextClassRef>
         urn:oasis:names:tc:SAML:2.0:ac:classes:PasswordProtectedTransport
       </saml:AuthnContextClassRef>
     </saml:AuthnContext>
   </saml:AuthnStatement>
   <saml:AttributeStatement>
     <saml:Attribute
       xmlns:x500="urn:oasis:names:tc:SAML:2.0:profiles:attribute:X500"
       x500:Encoding="LDAP"
       NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:uri"
       Name="urn:oid:1.3.6.1.4.1.5923.1.1.1.1"
       FriendlyName="eduPersonAffiliation">
       <saml:AttributeValue
         xsi:type="xs:string">member</saml:AttributeValue>
       <saml:AttributeValue
         xsi:type="xs:string">staff</saml:AttributeValue>
     </saml:Attribute>
   </saml:AttributeStatement>
 </saml:Assertion>
```

Figure 2.12: Example SAML assertion document.

```
{
  "iss":"joe",
  "exp":1300819380,
  "example/value":true
}
```

(a) JWT object

```
eyJpc3MiOiJqb2UiLCJleHAiOjEzMDA4MTkz
ODAsImh0dHA6Ly9leGFtcGxlL2lzX3Jvb3Qi
OnRydWV9
```

(b) Encoded token

Figure 2.13: JWT token in plain and encoded form. Here the `iss` and `exp` are registered claims defined by the standard but the `example/value` is a private claim defined by the user.

keys (HS256, HS384 and HS512) and `RSASSA-KCS1-v1_5` for public keys (RS256, RS384 and RS512) (JM15, M15).

2.2.5.2 JWE

JSON Web Encryption (JWE) tokens are encrypted JWT tokens, enforcing authenticated encryption (integrity and confidentiality) (JMH15, MH15).

2.2.6 Keycloak

Keycloak is a library and server to simplify the implementation of authentication and authorization systems (key, ey). It supports standards such as OAuth 2, OpenID Connect, SAML 2 (for authentication), and UMA 2.0 among others.

It is distributed as a web server that provides a dashboard for user login and configuration management, including managing access permissions by resource owners. It supports integration with third-party IdPs using OIDC for authentication. It also includes client libraries for various targets, such as java servers and javascript SPAs.

2.2.7 ORCID

As described on the website (orcb, rcb): "ORCID's vision is a world where all who participate in research, scholarship, and innovation are uniquely identified and connected to their contributions across disciplines, borders, and time.". It is a non-proprietary alphanumeric code that is assigned and used to uniquely identify scientific and academic authors.

Example `ORCID: 0000-0002-1825-0097` or the equivalent `https://orcid.org/0000-0002-1825-0097`.

ORCID supports OpenID Connect and implicit OAuth 2 (orca, rca) for use in authentication and in providing information about researchers, making it possible to use ORCID as a federated identity provider.

```
// header
{
  "alg": "HS256",
  "typ": "JWT"
}
// payload
{
  "iss":"joe",
  "exp":1300819380,
  "example/value":true
}
// signing symmetric key
// 1ac4a7c44258310fe93091c43fa94ddc
```

(a) JWT object

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXV
CJ9.eyJpc3MiOiJqb2UiLCJleHAiOjEzM
DA4MTkzODAsImV4YW1wbGUvdmFsdWUiOn
RydWV9.PBkuDG3hqxqCNSrxI3c3H1vphm
y5JM75bYiX_7ZuHlA
```

(b) Encoded token

Figure 2.14: JWS token in plain and encoded form. The algorithm used for signing is `HMAC SHA-256`,

2.2.8 EGI Check-in

EGI Check-in works as a central hub to aggregate multiple Identity Providers (including social media and ORCID (Fou17, ou17)) to provide an abstraction over them to allow applications registered with EGI to be able to consume any of the supported IdPs (egia, gia). It features a user registration portal for account-linking and it combines claims about the user from the various IdPs.

EGI check-in supports OpenID Connect for client integration (egib, gib).

2.3 Evaluation of technologies

The OAuth 2 protocol is a widely used standard for authorization but is not applicable for the authorization problem of this thesis because it requires direct user interaction with the resource owner, and the client does not act on behalf of the resource owner but on behalf of the researcher while OAuth assumes the client's user and the resource owner are the same.

UMA 2 which is an extension of OAuth 2 and attempts to remedy the issues mentioned above is a good candidate for implementing the authorization system, and while it is still a relatively new protocol and some vulnerabilities might be uncovered it provides the important feature of separation of client's user and resource owner.

The OpenID Connect protocol along with EGI Check-in is a good candidate for providing researchers with a single sign-on solution. OpenID Connect is an open standard, that while still in the final stages of standardization is widely used, and has been reviewed by the developer community for flaws (MMK+15, MK+15). It allows the holder of the ID token to prove that he is the person described in the token. Given that ORCID is widely used by researchers it should ideally be supported as an authentication method, which is already provided by the EGI Check-in IdP since it is a solution for aggregating various researcher identification methods like ORCID and social media, allowing a researcher to be assigned a unique ID that can be used to assign permissions to a researcher. Supporting EGI Check-In will also, by proxy, support various other

IdPs and therefore reduce the integration and configuration workload. OAuth 2 itself can also be used for single sign-on and is used successfully by big corporations like Facebook, Google, and Twitter, and while technically it can be implemented securely, frequently it is not (HHH+18, HH+18) since OAuth is designed for authorization and not authentication, while OpenID Connect is designed for authentication which should make it harder to implement insecurely (MMK+15, MK+15).

SAML 2 is a standard that can provide both authentication and authorization, but in comparison to OIDC in terms of authentication, it has some disadvantages such as lacking some important features (such as user consent) and not being lightweight (NJ17, J17). The main advantage of this protocol is that it has been widely used and there was time to uncover any vulnerabilities while OIDC is still a relatively new protocol.

JWT tokens are used in the OpenID Connect ID tokens (as JWS) and are a good candidate for protecting endpoints (essentially "passwords" for accessing resources) in a distributed system (EMWY17, MWY17), since in their signed form they cannot be modified by an adversary and contain in themselves claims that can be made about the user and other metadata like the token expiration time, token ID, etc.

Keycloak provides a server that implements the OpenID Connect protocol for authentication, handles account management (like login, registration), provides a permissions dashboard for resource owners to manage their resources, and exposes a UMA enabled OAuth authorization server interface. This can be useful for speeding up development while also making accounts management more secure since this authorization server has already been validated by deployment experience from multiple users, while a solution that involves "reinventing the wheel" of accounts management would inevitably have some bugs. This can be useful for managing resource owner accounts and the permissions they set for researchers.

Chapter 3

Problem

In this chapter, the problem space is explored along with strategies for validating a possible solution to the problem.

3.1 Goals

The goal of this thesis is to design an access control system with federated authentication for health researchers, using widely used authentication services, to protect access to DNA repositories and to design a way for resource owners to specify which resources are accessible by which researcher. Existing repositories are to be protected by an access control system based on this design.

This goal can be subdivided into:

- Allow a researcher to login using a federated authentication system like ORCID.
- Allow a researcher to access a resource to which he was given access, or to be able to request this access from the resource owner.
- Protect existing health data repositories such as sciReptor and iReceptor Turnkey.
- Allow a resource owner to manage access to his resources by researchers.
- Specify different access levels that a researcher and resource can have, where a researcher needs to have the same or above access level to access a resource. A public access level must be defined for information that anyone can access.
- Allow integration of the solution with a potential search engine capable of searching multiple repositories.

While no system is completely secure, given that the data the access control system is to protect is very sensitive, the design and implementation must be as secure as possible.

From these goals, some open-ended questions are raised which need some elaboration and consideration, which will be answered in the next chapter, such as:

- What are some appropriate access levels for resources, e.g. what information should encompass a public access level, and which others access levels should there be?
- How should search engine integration work with repositories that might expose different, incompatible APIs?
- Who should be a resource owner, and how should he be assigned that role?
- How should a user be identified as a researcher?

3.2 Research Strategy

This dissertation takes an engineering approach to the problem and so the research strategy consists of evaluating the technologies presented for building an access control system, such as UMA, OpenID Connect, SAML, among others, and implementing an access control system with these to evaluate their effectiveness and whether they allow the implementation of the functional requirements.

3.3 Problem Statement

Healthcare data is very sensitive and it makes sense to be able to restrict access to it to only certain approved users, though in the current landscape with the prevalence of the Internet and the ease with which data can be transmitted through it, many data repositories, such as sciReptor, immundeDB and iReceptor Turnkey, which expose an HTTP API, don't provide such a system for managing access or filtering requests by users and so certain personal information may be exposed to the public, including to third parties that should not have access. Additionally, new legislation such as the EU's GDPR makes this model of publicly accessible information unviable, and therefore an access control system is needed to manage access to healthcare information.

3.4 Expected Results

The solution's goal consists of a user being able to login using a federated authentication system and to request access to a resource. When access is granted the user would then obtain the resource using an HTTP API such as the ADC API.

Some repositories don't implement the ADC API and adding support for it can be expensive, so in these cases access control can be performed on the resource server level, while some other repositories implement the ADC API, in which case access control should be done at a layer above the resource server, to allow code re-use.

For all repository types, a web dashboard is needed to allow the management of access to resources.

3.5 Validation Strategy

In terms of validation, three approaches are taken:

- Validation of functional requirements that are present for the access control system, which are listed in the following chapter, such as, for example, a researcher being able to log in and access resources. These have a simple validation strategy consisting of checking whether the functions can actually be performed.
- Validation of the access control system by the users of the repositories which this system is supposed to protect, such as the users of the iReceptor Turnkey project. The input of the researchers using the repositories, of the admins managing the data, or of the DevOps engineers integrating the system with their repository, is important to know whether the system is useful to these users.
- Validation by domain experts, such as experts in HTTP security or the ADC API, of the solution and the architecture chosen.

Chapter 4

Healthcare Repositories Multi-layered Security

In this chapter, the solution requirements are listed along with some approaches to a solution.

4.1 Requirements

Three user roles are identified which interact with the system:

- Researcher: the user that will access the information
- Resource Owner: the user that owns resources in the resource server and can grant or revoke permissions to researchers to access his resources, using different access levels.
- Admin: the user responsible for managing the repository and assigning ownership of resources to resource owners.

Resource Owners and Admins would interact with the permissions management web dashboard component, while researchers would mainly interact with the resource server, authentication system, and the search engine (if available).

4.1.1 Resource Owners

The owner of the data needs to be able to specify which researcher has access to his resources and with which access level. This should be done through a web page dashboard to which the resource owner has access through a login page.

The resource owner as used until now is the dashboard user responsible for giving or revoking access, of either access level, for a study to a researcher. A resource owner can manage multiple studies. There can be multiple resource owners, with owners entering and leaving the system, each

managing permissions for a set of one or more studies. Multiple resource owners can manage the same study.

4.1.2 Researchers

The researcher needs to be able to authenticate himself with the system using a third party server (federated authentication) that the system needs to be able to understand and validate. The authentication system must support Identity Providers used frequently by researchers such as ORCID or EGI Check-In.

A researcher can access resources in the repository after being validated by the access control system.

A researcher could use a search engine to search through single or multiple repositories implementing the updated ADC interface, accessing resources to which he was granted access by the respective owners, therefore the authentication system needs to be flexible enough to allow this use case.

4.1.3 Repositories

The access control system (authorization) needs to be able to protect multiple repositories each exposing an HTTP API which might be incompatible with the ADC API specification, but with some commonalities. Ideally, the authentication and authorization layer should be decoupled from the repository implementation to allow it to be reused between the different repositories.

The repositories that are considered for protection by the project are:

- immuneDB: can be described as "a database-backed system to analyze and store large amounts (terabytes) of high-throughput B-cell receptor (BCR) and T-cell receptor (TCR) data" (Lab18, ab18). It does not implement the ADC API, but the exposed HTTP API has the concept of studies, subjects (related to repertoires), and sequences (related to rearrangements).

- iReceptor Turnkey: has the goal "to make it easy for a research group to download and install the software, curate their data into their repository, and then integrate that repository into the AIRR Data Commons network of repositories" (ireb, reb). It exposes a mostly complete ADC API.

- sciReptor: can be described as "a toolset to process, store and analyse single-cell level immunoglobulin (Ig) and T cell repector (TCR) sequence data." (sci, ci). It exposes a minimal ADC API with some similarities and differences to the complete API with a lot of features not implemented, though the API is originally based on the "processing and analysis of antigen receptor repertoire sequencing data at single-cell level" tool as described by (IAWB16, AWB16) so API compliance with ADC is not the main focus of this project.

There are also other repositories not mentioned here that handle DNA data such as VDJServer (VDJ, DJ), but which are outside the scope of this thesis so far.

Some repositories don't fully or correctly implement the ADC API, as exemplified by sciReptor, or might have different return values for their endpoints, or even implement a completely different API, so a general authorization solution might not be applicable in some cases, where the authorization implementation might need to be coupled with the repository implementation, or ideally, the repositories should be updated to expose a correct ADC API. Additionally, a repository might expose an API in addition to the ADC API, such as iReceptor Turnkey, which needs to be taken into account.

In the more general case where the authorization layer is only coupled to the ADC API, and not the underlying repository implementation, the problem of synchronizing the state of the dashboard web server and the state of the underlying server is introduced, that is, the problem of making sure the dashboard web server has the same studies, repertoires, rearrangements as the underlying server, so as to not introduce inconsistencies.

ADC API Requirements

As mentioned, some repositories implement the ADC API open standard for DNA data. It is an HTTP+JSON API and is currently in version v1 and supports repertoires, rearrangements, searching based on JSON fields and some statistics related to counting fields for a search (facets).

There are proposals to extend this API as part of the iReceptor+ project with no currently final version for the changes, which involve adding support for clonotypes and statistics. As a result of the new resource types and endpoints to be added the access control system needs to be flexible enough to be able to protect with ease the evolving API.

The API defines three main resource types: studies, repertoires, and rearrangements. A study is the top-level resource and each study may contain multiple repertoires. A repertoire may contain multiple rearrangements. A study can be thought of as a medical study containing metadata about a group of subjects. A repertoire represents a single patient and contains information like diagnosis and personal information like age, gender. A rearrangement can be thought of as aggregated DNA information, produced by some DNA processing software.

It should be noted that the number of rearrangements in a repository can be very large, in the order of gigabytes or more, so an access control solution needs to be able to handle high volumes of data being returned.

Researchers are to be granted access to, on a per-study basis, one of two access levels:

- Level 1: statistics about the study (more general and permissible case): statistics about repertoires and access to repertories themselves which provide some aggregate information about the study and subject.

- Level 2: the raw DNA sequences, as part of the repertoire resource and of all the rearrangements in the study.

One access level contains all the privileges of the previous level, i. e. a researcher with access level 2 can also access resources from level 1. The extensions to the ADC API might require

adding additional access levels to be able to address clonotypes and the new statistics endpoints and to grant access based on, for example, a per-repertoire basis.

A researcher can be granted access to multiple studies with any of the access levels.

Additionally, some ADC endpoints need to be accessible by any researcher (public endpoints), such as endpoints or requests that only return server health-checks, or return the list of resource IDs without any additional information, IDs which can be used to retrieve individual resources, which are protected, or information about the studies involved in the data such as the ID, title, types, etc. Some additional information might be deemed public and the system needs to be flexible enough to allow specifying which data can be accessed by any researcher.

4.2 High-level Architecture

The tools and technologies chosen for the solution are:

- OpenID Connect (OIDC): to support researcher federated authentication, using Identity Providers such as ORCID.
- UMA 2: to allow asynchronous authorization control for resources, where the resource owner is different from the resource user.
- Keycloak: to act as an authorization server in the UMA sense and as a consumer of OIDC tokens for authentication. Keycloak can handle all of the authentication needs of the solution so that the main focus is on designing the authorization system. Keycloak also provides a ready-made permissions management dashboard, though with some limitations, and an admin panel for managing Keycloak and UMA resources (such as name, description, etc).

And as such, as can be seen in figure 4.1, and as mentioned previously, several components compose the solution:

- A permissions management dashboard. This component is responsible for managing permissions of researchers to resources, and these need to be managed by their respective resource owners. As mentioned, Keycloak supplies a web dashboard, allowing a resource owner to log in on a webpage, see which researchers have requested access to resources he owns (in the UMA flow sense, where when access is denied to emit an RPT token a request is placed on the system in the Keycloak database), and grant or revoke access with scope granularity (see below for a translation of UMA scopes to access levels). Additionally, an owner can pre-emptively grant access to a researcher without it being requested in the UMA sense. A researcher needs to have an account in Keycloak to be able to be granted access, though Keycloak also supports OpenID Connect based logins, which associate an identity with an account in Keycloak.
- A login panel for researchers. A researcher can use an Identity Provider that exposes an OpenID Connect interface such as ORCID to log in, since Keycloak can consume that ID

token to create a local account for the researcher associated with the ID token, to allow him to be granted or denied access to resources. A researcher can also have a regular user profile in Keycloak, but this needs to be created by an admin in the Keycloak sense, and this login flow would behave the same way as the login through a third party OIDC IdP.

- A UMA authorization server: this job would be filled by Keycloak since it exposes a UMA 2 interface that can be used by the Middleware server to enforce access control.

- Middleware Server: this would act as an access control enabled reverse proxy for repositories that expose an ADC API. This component would handle all of the access control related functions, such as requesting UMA tickets, processing RPT tokens, filtering requests and responses from the repositories, etc. Some repositories have an internal API incompatible with the ADC API, and in those cases, the middleware server is not applicable as a drop-in solution. Each server is, ideally, the exact same and self-sufficient and only coupled to the ADC specification. This component is the main focus of this thesis.

- The resource server (repository). As mentioned the repositories should ideally at least implement a minimally correct ADC API to allow the use of a middleware server that can handle all of the access control functions for them, which has the advantage of the solution being reusable between multiple repositories. In this case, a researcher that wants to access a resource server repository, be it iReceptor Turnkey, sciReptor, or any other exposing the ADC API, has to first go through each of the middleware servers.

  Some repositories might not follow the ADC API, or might expose other APIs in addition to ADC, such as iReceptor Turnkey's, so for these cases, the authorization is done by the underlying repository which has the disadvantage of code repetition since the integration with UMA needs to be done individually for each repository.

Compared to its unprotected version, an API protected with UMA 2 has worse performance, given that each request (at least initially) needs to be repeated twice, once to get the permissions ticket, and then again with the RPT token to get the resource. Additionally, the server implementing the flow and the researcher need to make additional requests to the UMA authorization server (Keycloak) to generate the permissions ticket, exchange it for an RPT token, and introspect it. However, the first two requests can be unnecessary if the RPT token is reused and the last request can be implemented by inspecting and validating, using the public key of the authorization server, the JWS-encoded RPT token (if the authorization server supports it).

The middleware server for the ADC repositories should ideally be hosted alongside with Keycloak, though it is also possible to host Keycloak separately and allow multiple repositories to be managed by a central Keycloak server. This is not recommended, since if there is a security breach in the central Keycloak instance all of the repositories are affected.

A researcher can also use a search engine (gateway in the figure), which in practice works as a user-friendly front-end for the researcher, to login and search in multiple repositories. Each search would go through the middleware server the same way as accessing the resource server directly

in the ADC case. For non-ADC repositories, the search engine might need to have adapters for the specific API to display a uniform UI, which might not be possible if there are contradictions between the ADC API and the internal API concepts.

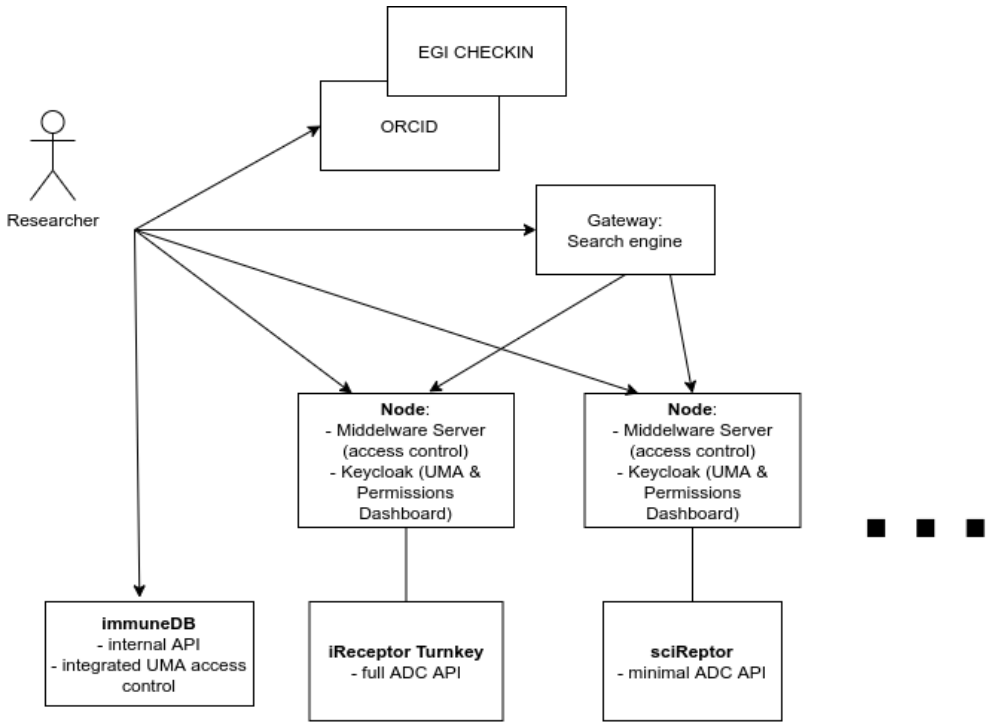


Figure 4.1: Proposed solution architecture. Here each node contains both the authorization server, dashboard server and middleware server authorization layer.

4.3 Federated Authentication

As mentioned before, Keycloak allows integration with third-party OpenID Connect Identity Providers, such as ORCID. In this case, Keycloak would essentially act as a proxy.

The login flow can be resumed in Figure 4.2. A login request is first made by the researcher's client, which redirects the researcher to the Keycloak login web page, the user selects the desired OIDC IdP and Keycloak, in turn, redirects the researcher to the selected IdP, which when logged in, redirects back the ID token to Keycloak which consumers it and generates a new token and redirects it back to the researcher's client for use with the UMA flow.

As mentioned the researcher can instead also just use a regular Keycloak account (which needs to previously be created by a Keycloak admin) since the token emitted by Keycloak on login for use with the UMA flow is the same type as the token emitted with the help of a third-party IdP.

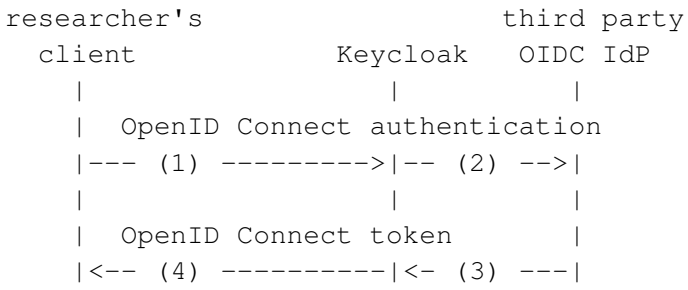


Figure 4.2: Flow for federated login using Keycloak. The flow ends with a login token emitted by Keycloak. Steps 2 and 3 can be omitted if using a local Keycloak account.

4.4 Resource-server Security

Access control can be done either on the repository level, that is, UMA access control flow integrated with the repository implementation and code, or as a separate component, that is, UMA access control handled by a proxy server (middleware server) that handles all of the access control functionalities and validates and filters the requests and responses by making use of request forwarding.

Using a middleware server makes sense when multiple repositories implement the same API, such as in the case of the ADC API, allowing the same component to be reused between multiple resource servers, but it also has some disadvantages as discussed below. For repositories that have a unique internal API, integrated access control in the repository might make more sense, since there can be performance benefits, which matters in the case of DNA data, and software process benefits from having vertical integration.

In either case, the repository or the middleware server would implement the UMA 2 flow for managing access control.

From a researcher's perspective the following steps need to be followed to access a resource, as can be seen in figure 4.3, whether for an ADC API or otherwise:

1. The researcher's client authenticates the researcher using the OpenID Connect protocol with Keycloak (which may involve authenticating with a third party OIDC IdP) and is issued a login token. As defined by the specification, the login token contains an ID to uniquely identify the researcher and some additional claims that prove that the token holder is the researcher described in the token and some claims specific to Keycloak.

2. The researcher's client attempts to access a resource by sending a request without an RPT token to the resource server (access is denied).

3. The resource server sends a request to the authorization server (Keycloak), specifying the resource IDs (associated with a study resource) and scopes (associated with an access level) and is returned a permission ticket. The middleware server is responsible for translating access to a request into the corresponding resources and scopes. The authorization server only understands UMA resource IDs which are different from the study IDs, so the resource

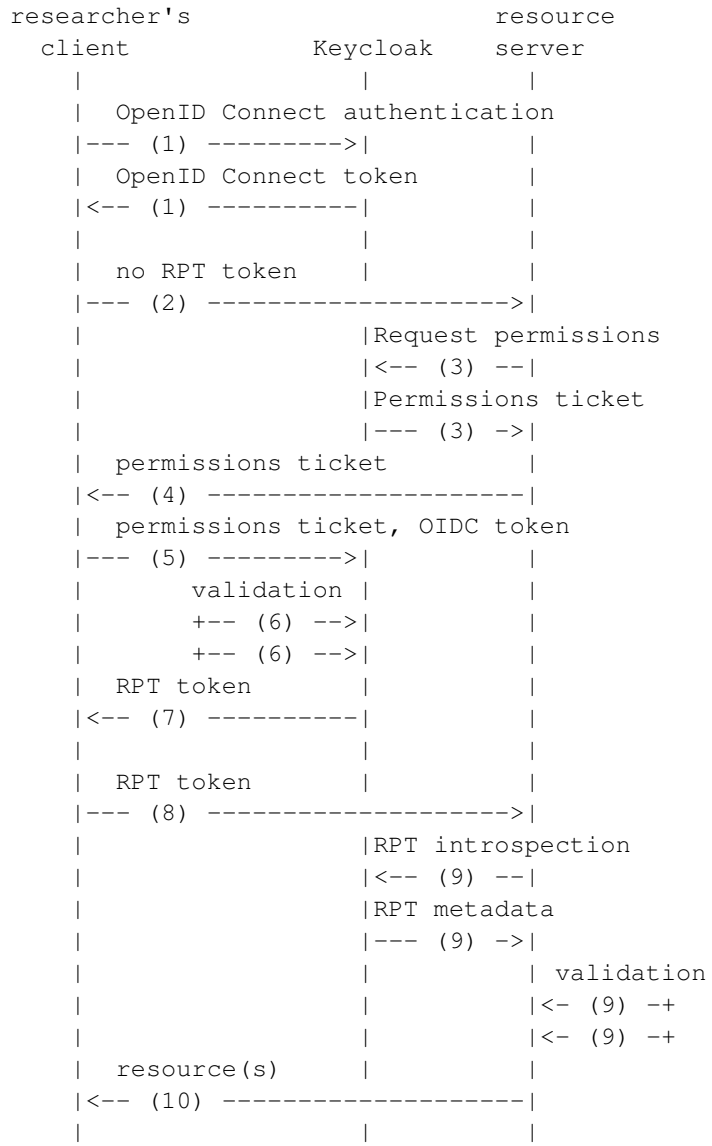


Figure 4.3: Flow for accessing a resource from a researcher's perspective. Keycloak acts as both an OIDC IdP and an UMA authorization server. The resource server can either be the repository itself or a middleware server which filters requests for the repository.

server must keep a mapping between each study's ID and the corresponding UMA resource ID, which can be obtained when creating the resource in Keycloak.

4. The resource server denies access and returns the location of the authorization server along with the permission ticket from the previous step.

5. The researcher's client requests an access token (RPT token) from the authorization server by sending the permission ticket from the previous step, the OpenID Connect token from step 1, and optionally any additional scopes the researcher wants access to.

6. Keycloak validates that the ID token is valid and finds the resources and their scopes that the researcher has been given permission to by the resource owner. It then performs an intersection between the granted resources and scopes and the requested resources and scopes as specified in the ticket: first, the resources IDs are intersected, and then for each resource, the scopes are intersected. For any resource-scope pair that is requested but for which access is denied, Keycloak places it for asynchronous review by the resource owner, which can grant or deny access to each resource-scope pair requested. A researcher can in this way request access to resources from the resource owner by simply trying to access them.

7. Keycloak returns an RPT token which is the resulting intersection of the resources and their scopes, that allows the client to access these resources. If the intersection is null no token is returned.

8. The researcher attempts to access the same resource from step 2 using the new RPT token.

9. The resource server validates the token and obtains the associated scopes and resources for the token from the authorization server, which also involves translating the UMA resource IDs to the study IDs, and validates that the RPT token grants access to the requested resource.

10. If validation passes, the resource is returned, which might involve forwarding the request if the resource server is a middleware server, or just simply returning either the full or filtered resource if the resource server is the repository itself. If validation fails, an HTTP error is returned.

Step 1 is simply using the OpenID Connect protocol to allow a researcher to use federated login, while steps 2 to 10 are simply the flow of the UMA 2 protocol.

If the researcher wants to access resources other than the ones he specified in step 2, steps 2 to 7 are repeated unless the user has an RPT token from a previous exchange that is still valid and grants access to the resource, in which case only steps 8 to 10 need to be executed.

In step 10 resource filtering is mentioned, given that the APIs considered return JSON objects (resources), filtering here means that a subset of the JSON object is returned (a subset of the fields of the resource).

4.4.1 Repository-level Access Control

In this approach, the UMA 2 flow and access control are implemented directly in the repository. This is the approach taken to protect immuneDB by the project's collaborators and would be the approach taken to secure the additional API exposed by iReceptor Turnkey.

This approach has some advantages, namely:

- Performance: compared to a middleware server solution, this approach can have the advantage of better performance (faster requests), which matters since DNA sequence data (which all repositories expose) can be very large. It achieves this performance since the repository can directly inject into the database queries the permissions taken from the RPT token and only return the accessible information, from the DB layer to the researcher. Additionally, a middleware server introduces some overhead since it is another layer on top of the backend.

- Flexibility: some API designs might make it impossible to protect them using a middleware server solution, since for example nowhere in the request or the response might be specified a study ID for access control validation by the middleware. This allows greater flexibility in the design of the API.

And some disadvantages, namely:

- Code reuse: for repositories that implement the same API, such as sciReptor and iReceptor Turnkey, implementing the same access control for each repository leads to code duplication and may lead to bugs and security issues which might not be present if using a single well-developed and tested solution.

- Code Coupling: the repository code would have to become coupled with UMA concepts, to be able to implement access control, and additionally would have to store the mapping between the resource IDs protected and their corresponding UMA resource IDs. Although this might be an advantage since it would make security considerations a clearer issue to the repository developers.

In terms of access flow from a researcher's perspective, it would be the same as described in figure 4.3, except that step 10 would only involve returning the filtered resource(s) according to the RPT token received.

4.4.2 Middleware Server Access Control

In this approach access control is pushed up a layer from the repository into its own server, the middleware server. This server is then responsible for correctly implementing the UMA flow, and access control, by validating and filtering researcher requests, forwarding them to the underlying repository (which might involve modifying the request), and validating and filtering the response. This flow can be summarized in figure 4.4.

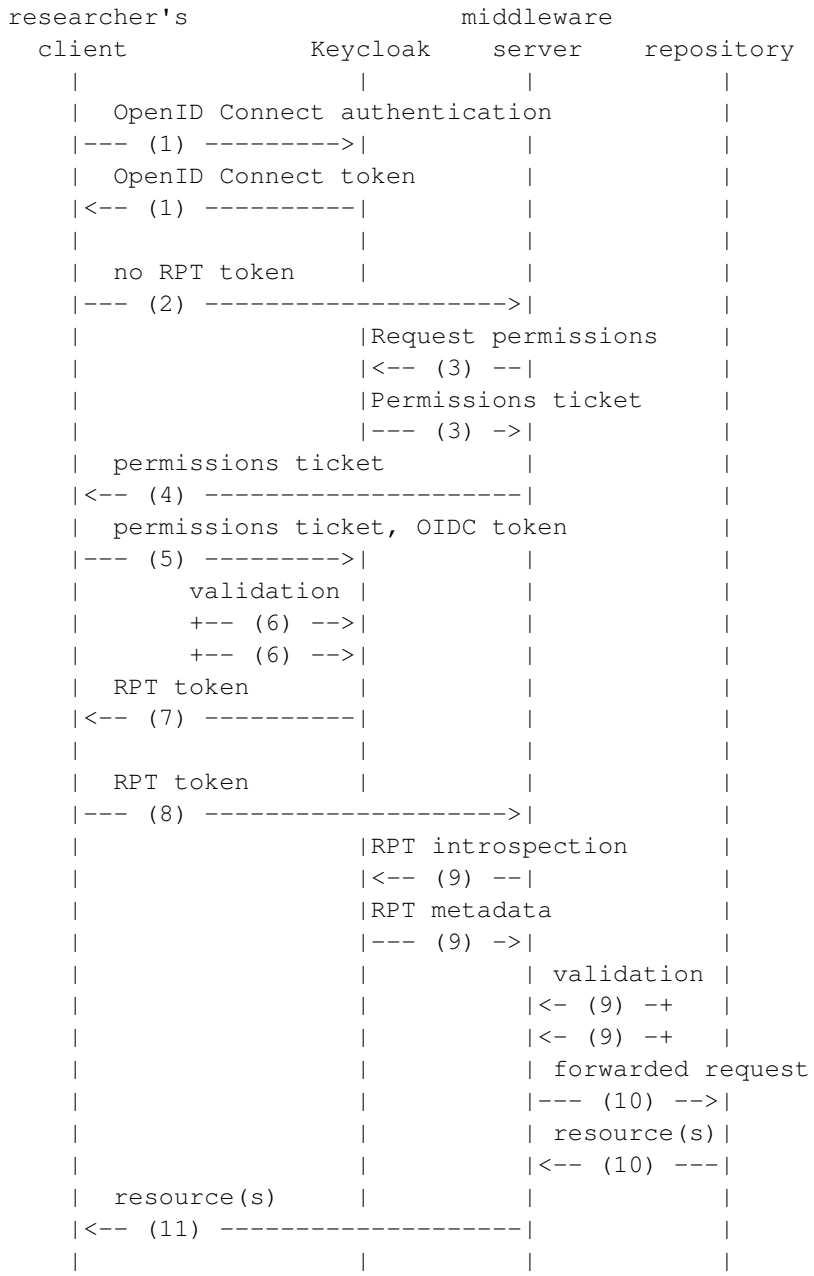


Figure 4.4: Flow for accessing a resource from a researcher's perspective with additional details about the internal workings of the middleware server.

From an API agnostic researcher's perspective, the flow to access a resource is the same as in the case of the repository-level access control but the internal requests used change. The main differences between this flow (figure 4.4) and the general flow (figure 4.3) are:

1. A step is introduced (step 10) which is the request the middleware server sends to the repository to obtain the requested resource(s). This will only happen if all of the validations pass, and the request might be modified from what the researcher requested to fulfill the access control restrictions as indicated in the RPT token.

2. Step 10 from the general flow is now step 11 here, and it might involve filtering the response if the resources returned by the repository contain fields belonging to UMA scopes that the user does not have access to.

This approach has some advantages and disadvantages, which are the mirror image of the advantages and disadvantages of the repository-level approach. Advantages:

- Code Reuse: if there is more than 1 repository implementing the same API the same middleware server solution could be used for all of them, removing code duplication.
- Code Decoupling: by making the middleware manage all of the access control related functions the repository implementation doesn't need to be aware of any UMA concepts.

And disadvantages:

- Performance: the performance overhead of the requests, as discussed before can become worse. But a careful design of the API to be more UMA aware, as discussed below, can make this overhead lower.
- Flexibility: by making the access control become an external component the repository developers have less flexibility in adding new endpoints or extensions to the existing endpoints if this is not taken into account by the middleware server. Additionally, an inconsistency between the assumptions made by the middleware server and the repository for an endpoint could cause security issues, like data leaks.

This approach is mainly useful when there is more than one repository implementing the same HTTP API, which is the case of the ADC API with the sciReptor and iReceptor Turnkey repositories.

ADC API

As mentioned previously, and as can be seen in the official API documentation (mia, ia), the AIRR's ADC API, currently version 1, exposes the concept of studies, repertoires, and rearrangements, though the studies are embedded in rearrangements. A study can contain many repertoires and a repertoire can contain many rearrangements.

Endpoint	**Type**	**Method**
(1) /v1	Service status	GET
(2) /v1/info	Service information	GET
(3) /v1/repertoire/{id}	Retrieve a repertoire given its repertoire_id	GET
(4) /v1/repertoire	Query repertoires	POST
(5) /v1/rearrangement/{id}	Retrieve a rearrangement given its sequence_id	GET
(6) /v1/rearrangement	Query rearrangements	POST

Table 4.1: ADC API overview, current standard (v1).

The endpoints that define the API, which can be summarised in table 4.1, have the following characteristics:

- Endpoints 1 and 2 return metadata information, like status check, the information header, and do not expose any sensitive information, see figure 4.10.
- Endpoints 3 and 4 return repertoire resources and endpoints 5 and 6 return rearrangement resources. For each pair of endpoints the resource fields that can be returned are the exact same. Endpoints 3 and 5 returns only one resource while 4 and 6 can return any amount of resources in list form or a counting of a field (facets).
- Endpoints 3 and 5 return the resources as specified by their ID. Endpoint 3 returns the repertoire resource given its repertoire ID (field `repertoire_id`), see figure 4.5, and endpoint 5 returns the rearrangement resource given its sequence ID, see figure 4.6, (`sequence_id` is the rearrangement's ID even though there already is the field `rearrangement_id` but which was deprecated in favour of the sequence ID field as rearrangement's main ID). The HTTP response is in JSON format (JSON object). These endpoints will be called individual endpoints from now on.
- Endpoints 4 and 6 allow a user of the API to search the repository for resources matching the user's query, see figures 4.7, 4.8 and 4.9. The query is sent in the HTTP body in JSON object format. The request query JSON object fields allowed are (any one of the fields can be omitted from the request):
  - `"from"`: A positive integer, specifies the amount of resources to skip, useful for paging, see figure 4.8. Defaults to `0`.
  - `"size"`: A positive integer, specifies the number of resources to return, skipping the rest, useful for paging, see figure 4.8.
  - `"format"`: A string, can be either `"json"`, the default, which specifies that the data should be returned in JSON format, or `"tsv"` which specifies that the data should be returned in TSV format (tab separated values, similar to CSV), see figure 4.8.
  - `"fields"`: A JSON list of strings, specifies the set of resource fields to be returned which would cause the repository to return only the requested fields for the resource and not all of the resource's fields, see figure 4.7. For each repertoire and rearrangement resources there are specific fields that can be requested.
  - `"include_fields"`: A string, which can be one of `"miairr"`, `"airr-core"` or `"airr-schema"`, see figure 4.8. Performs the same function as the `"fields"` query parameter but instead of specifying individual fields each value in this parameter specifies a pre-selected set of fields to be filtered. Both this parameter and the `fields` parameter is additive, that is if this parameter specifies a set of resource fields and the `"fields"` parameter specifies another set the returned resource fields are the union of the 2 sets. `"miairr"` is the smallest set, and is contained in the `"airr-core"` set, which is contained in the `"airr-schema"` set which contains all of the AIRR fields for the resource. For each repertoire and rearrangement resource the values of this parameter specify a specific set of fields.

- `"filters"`: A JSON object, specifies a search expression to match the resources against, see figure 4.7. The value is a filter object. A filter object consists of an operator (`"op"`) string, which specifies the query type and a content field (`"content"`) which can contain a field string (`"field"`), and a matcher value (`"value"`, values can be numbers, booleans, strings or lists of strings), to run the operator against or a JSON list of other filters objects. Valid operators are:
  - `"="`, `"!="`, `"<"`, `"<="`, `">"`, `">="`, `"contains"`: These operators will match a resource's field against a provided single value for the field.
  - `"is missing"`, `"is"`, `"is not missing"`, `"not"`: These operators will check that the specified resource's field is present and not null or the inverse. Here the `"value"` key is omitted from the `"content"` object.
  - `"in"`, `"exclude"`: These operators will match a resource's field against the provided list of values, whether for equality or inequality.
  - `"and"`, `"or"`: These operators perform a logical operation between filters allowing to use multiple filters in the search query. The `"content"`'s value is a JSON list of filter objects.
- `"facets"`: A string, the resource field, see figure 4.9. When this parameter is present the endpoint changes function and instead of returning a list of resources returns the counting statistics for the requested field, that is for the resources that match the query the values for the field are counted and the count for each value is returned. The request is invalid if in addition to this the `"fields"` or `"include_fields"` parameters are present. This field can be used along with the `"filters"` field to perform a search, but not with the `"from"` or `"size"` fields.

The repertoire resource contains nested objects and arrays of objects, so to be able to request these fields in the `"fields"`, `"facets"` and in the filters objects in the `"filters"` parameter the user can denote the path to the nested field by separating the composing fields using a dot, for example, `study.study_id` would access the repertoire's `study` top-level JSON object key and then from the nested object value would access the `study_id` key, for arrays each of the list's object element's field would be considered.

These endpoints will be called search endpoints from now on.

It should be noted that in the examples presented here, the API version is v1.3 because it was recently officially renamed to v1 and the repository used had not yet updated the version designation.

As mentioned previously the resources returned by the ADC endpoints are of 2 types:

- Repertoire: represents a subject (e.g. a person) and contains information about the study related to the repertoire, some DNA information, and some metadata. A lot of fields compose this resource, some of them nested inside documents, but the most important are: `repertoire_id` which is the resource's ID, `study`'s `study_id` which is the study ID to which

```
{
  "Info": {
    "Title": "AIRR Data Commons API",
    "description": "API response for repertoire query",
    "version": 1.3,
    "contact": {
      "name": "AIRR Community",
      "url": "https://github.com/airr-community"
    }
  },
  "Repertoire": [
    {
      "repertoire_id": "5e53de7f9463684866be6092",
      "repertoire_name": null,
      "repertoire_description": null,
      "study": {
        "study_id": "PRJNA330606",
        "study_title": "The Different T-cell Receptor
            ↪ Repertoires in Breast Cancer Tumors,
            ↪ Draining Lymph Nodes, and Adjacent Tissues
            ↪ ",
        "study_type": {
          "label": "Case-Control Study",
          "id": "C15197"
        },
        "study_description": "Cancer Study",
        "inclusion_exclusion_criteria": "",
        "grants": "",
        "collected_by": " liuxiao@genomics.cn,
            ↪ vascular@fmmu.edu.cn",
        "lab_name": "Department of Vascular and Endocrine
            ↪ Surgery",
        "lab_address": "Beijing Genomics Institute (BGI)
            ↪ Xijing Hospital, Fourth Military Medical
            ↪ University Medical University, Xi'an,
            ↪ China",
        "submitted_by": "T. Wang, C.Wang, J. Wu, C. He, X.
            ↪ Liu",
        "pub_ids": "http://cancerimmunolres.aacrjournals.
            ↪ org/content/5/2/148.long",
        "keywords_study": [
          "contains_tcr"
        ]
      },
      "subject": {
        "subject_id": "BC0001A",
        "synthetic": false,
        "species": {
          "label": null,
          "id": null
        },
        "organism": {
          "label": "Homo sapiens",
          "id": "9606"
        },
        "sex": "Female",
        "age": "43 years",
        "age_min": 43,
        "age_max": 43,
        "age_unit": {
          "label": "year",
          "id": "UO_0000036"
        },
        "age_event": "",
        "ancestry_population": "",
        "ethnicity": "",
        "race": "",
        "strain_name": "",
        "linked_subjects": "",
        "link_type": "",
        "diagnosis": [
          {
            "study_group_description": "Case",
            "disease_diagnosis": {
              "label": "Breast Cancer",
              "id": null
            },
            "disease_length": "",
            "disease_stage": "ER3+, PR2+, LN ratio 0/16,
                ↪ Luminal B tumor subtype, ki67 20%",
            "prior_therapies": "",
            "immunogen": "",
            "intervention": "",
            "medical_history": ""
          }
        ]
      },
      "sample": [
        {
          "sample_processing_id": "5e53de7f9463684866be6092
              ↪ ",
          "sample_id": "BC0001A-1",
          "sample_type": "",
          "tissue": {
            "label": "Breast tissue",
            "id": null
          },
          "anatomic_site": "",
          "disease_state_sample": "Breast Cancer",
          "collection_time_point_relative": "Time 0",
          "collection_time_point_reference": "Blood
              ↪ withdraw1",
          "biomaterial_provider": "",
          "tissue_processing": "",
          "cell_subset": {
            "label": "CD4/CD8 T cells",
            "id": ""
          },
          "cell_phenotype": "CD3+ CD4+ CD8+",
          "cell_species": {
            "label": "",
            "id": ""
          },
          "single_cell": false,
          "cell_number": 12098941,
          "cells_per_reaction": null,
          "cell_storage": false,
          "cell_quality": "",
          "cell_isolation": "",
          "cell_processing_protocol": "",
          "template_class": "DNA",
          "template_quality": "",
          "template_amount": "",
          "library_generation_method": "PCR",
          "library_generation_protocol": "",
          "library_generation_kit_version": "",
          "pcr_target": [
            {
              "pcr_target_locus": "CDR3",
              "forward_pcr_primer_target_location": "FR3
                  ↪ region",
              "reverse_pcr_primer_target_location": "TRBJ "
            }
          ],
          "complete_sequences": "complete ",
          "physical_linkage": "nan",
          "sequencing_run_id": "",
          "total_reads_passing_qc_filter": 7272136,
          "sequencing_platform": "Illumina HiSeq 2000",
          "sequencing_facility": "",
          "sequencing_run_date": "",
          "sequencing_kit": "",
          "sequencing_files": {
            "file_type": "",
            "filename": "",
            "read_direction": "",
            "read_length": null,
            "paired_filename": "",
            "paired_read_direction": "",
            "paired_read_length": null
          }
        }
      ],
      "data_processing": [
        {
          "data_processing_id": "5e53de7f9463684866be6092",
          "primary_annotation": true,
          "software_versions": "SRA Toolkit 2.8.2-1",
          "paired_reads_assembly": "PEAR (Galaxy Version
              ↪ 0.9.6.0)",
          "quality_thresholds": "30",
          "primer_match_cutoffs": "",
          "collapsing_method": "",
          "data_processing_protocols": "Adaptors and
              ↪ primers removed and mixcr annotation of
              ↪ quality filtered sequences",
          "data_processing_files": [
            "SRR4084215_aa_mixcr.clns",
            "SRR4084215_aa_mixcr.vdjca",
            "SRR4084215_aa_mixcr_annotation_1000_lines.txt.
                ↪ gz",
            "SRR4084215_aa_mixcr_clones.txt",
            "SRR4084215_ab_mixcr.clns"
          ],
          "germline_database": "IMGT",
          "analysis_provenance_id": "1"
        }
      ]
    }
  ]
}
```


Figure 4.5: An example request to the endpoint `GET /v1/repertoire/id`. For example: `GET /v1/repertoire/5e53de7f9463684866be6092`, notice the `repertoire_id` field. All of the repertoire's fields are shown.

```
{
  "Info": {
    "Title": "AIRR Data Commons API",
    "description": "API response for repertoire query",
    "version": 1.3,
    "contact": {
      "name": "AIRR Community",
      "url": "https://github.com/airr-community"
    }
  },
  "Rearrangement": [
    {
      "sequence": "GAGCTCTCTGGAGCTGGGGG
          ACTCAGCTTTGTATTTCTGTGCCAGCAGCGTAGGG
          ACAGTTGGAAACACCATATATTTTGGAGAGGG",
      "sequence_aa": null,
      "rev_comp": null,
      "productive": true,
      "vj_in_frame": null,
      "stop_codon": null,
      "complete_vdj": null,
      "locus": "TRB",
      "v_call": "TRBV9*01",
      "d_call": "TRBD1*01",
      "d2_call": null,
      "j_call": "TRBJ1-3*01",
      "c_call": null,
      "sequence_alignment": null,
      "sequence_alignment_aa": null,
      "germline_alignment": null,
      "germline_alignment_aa": null,
      "junction": "
          ↪ TGTGCCAGCAGCGTAGGGACAGTTGGAAACACCATATATTTT",
      "junction_aa": "CASSVGTVGNTIYF",
      "np1": null,
      "np1_aa": null,
      "np2": null,
      "np2_aa": null,
      "np3": null,
      "np3_aa": null,
      "cdr1": null,
      "cdr1_aa": null,
      "cdr2": null,
      "cdr2_aa": null,
      "cdr3": null,
      "cdr3_aa": null,
      "fwr1": null,
      "fwr1_aa": null,
      "fwr2": null,
      "fwr2_aa": null,
      "fwr3": null,
      "fwr3_aa": null,
      "fwr4": null,
      "fwr4_aa": null,
      "v_score": 260,
      "v_identity": null,
      "v_support": null,
      "v_cigar": null,
      "d_score": 35,
      "d_identity": null,
      "d_support": null,
      "d_cigar": null,
      "d2_score": null,
      "d2_identity": null,
      "d2_support": null,
      "d2_cigar": null,
      "j_score": 135,
      "j_identity": null,
      "j_support": null,
      "j_cigar": null,
      "c_score": null,
      "c_identity": null,
      "c_support": null,
      "c_cigar": null,
      "v_sequence_start": null,
      "v_sequence_end": null,
      "v_germline_start": null,
      "v_germline_end": null,
      "v_alignment_start": null,
      "v_alignment_end": null,
      "d_sequence_start": null,
      "d_sequence_end": null,
      "d_germline_start": null,
      "d_germline_end": null,
      "d_alignment_start": null,
      "d_alignment_end": null,
      "d2_sequence_start": null,
      "d2_sequence_end": null,
      "d2_germline_start": null,
      "d2_germline_end": null,
      "d2_alignment_start": null,
      "d2_alignment_end": null,
      "j_sequence_start": null,
      "j_sequence_end": null,
      "j_germline_start": null,
      "j_germline_end": null,
      "j_alignment_start": null,
      "j_alignment_end": null,
      "cdr1_start": null,
      "cdr1_end": null,
      "cdr2_start": null,
      "cdr2_end": null,
      "cdr3_start": null,
      "cdr3_end": null,
      "fwr1_start": null,
      "fwr1_end": null,
      "fwr2_start": null,
      "fwr2_end": null,
      "fwr3_start": null,
      "fwr3_end": null,
      "fwr4_start": null,
      "fwr4_end": null,
      "v_sequence_alignment": null,
      "v_sequence_alignment_aa": null,
      "d_sequence_alignment": null,
      "d_sequence_alignment_aa": null,
      "d2_sequence_alignment": null,
      "d2_sequence_alignment_aa": null,
      "j_sequence_alignment": null,
      "j_sequence_alignment_aa": null,
      "c_sequence_alignment": null,
      "c_sequence_alignment_aa": null,
      "v_germline_alignment": null,
      "v_germline_alignment_aa": null,
      "d_germline_alignment": null,
      "d_germline_alignment_aa": null,
      "d2_germline_alignment": null,
      "d2_germline_alignment_aa": null,
      "j_germline_alignment": null,
      "j_germline_alignment_aa": null,
      "c_germline_alignment": null,
      "c_germline_alignment_aa": null,
      "junction_length": 42,
      "junction_aa_length": 14,
      "np1_length": null,
      "np2_length": null,
      "np3_length": null,
      "n1_length": null,
      "n2_length": null,
      "n3_length": null,
      "p3v_length": null,
      "p5d_length": null,
      "p3d_length": null,
      "p5d2_length": null,
      "p3d2_length": null,
      "p5j_length": null,
      "consensus_count": null,
      "duplicate_count": "",
      "cell_id": null,
      "clone_id": null,
      "sequence_id": "5e53dead4d808a03178c7831",
      "repertoire_id": "5e53de7f9463684866be6092",
      "sample_processing_id": "5e53de7f9463684866be6092",
      "data_processing_id": "5e53de7f9463684866be6092",
      "germline_database": null,
      "rearrangement_set_id": null,
      "v_gene": "TRBV9",
      "v_subgroup": "TRBV9",
      "j_gene": "TRBJ1-3",
      "j_subgroup": "TRBJ1",
      "d_gene": "TRBD1",
      "d_subgroup": "TRBD1"
    }
  ]
}
```


Figure 4.6: An example request to the endpoint `GET /v1/rearrangement/id`. For example: `GET /v1/repertoire/5e53dead4d808a03178c7831`, notice the `sequence_id` field. All of the rearrangement's fields are shown.

```
{
  "format": "json",
  "fields": [
    "repertoire_id",
    "study.study_id",
    "subject.diagnosis.
        ↪ study_group_description"
  ],
  "filters": {
    "op": "and",
    "content": [
      {
        "op": "in",
        "content": {
          "field": "repertoire_id",
          "value": [
            "395",
            "394",
            "5e53de7f9463684866be6092"
          ]
        }
      },
      {
        "op": "=",
        "content": {
          "field": "study.study_id",
          "value": "PRJNA248411"
        }
      }
    ]
  }
}
```

(a) Request body

```
{
  "Info": {
    "Title": "AIRR Data Commons API",
    "description": "API response for
        ↪ repertoire query",
    "version": 1.3,
    "contact": {
      "name": "AIRR Community",
      "url": "https://github.com/airr-
          ↪ community"
    }
  },
  "Repertoire": [
    {
      "repertoire_id": "394",
      "study": {
        "study_id": "PRJNA248411"
      },
      "subject": {
        "diagnosis": [
          {
            "study_group_description": "Case
                ↪ "
          }
        ]
      }
    },
    {
      "repertoire_id": "395",
      "study": {
        "study_id": "PRJNA248411"
      },
      "subject": {
        "diagnosis": [
          {
            "study_group_description": "Case
                ↪ "
          }
        ]
      }
    }
  ]
}
```

(b) Response

Figure 4.7: An example request to the endpoint `POST /v1/repertoire` with a user query. Notice that the repertoires returned don't have all their fields as specified by `"fields"` and that the repertoires returned all match the `"filters"` query.

```
{
  "size": 2,
  "from": 1,
  "include_fields": "miairr"
}
```

(a) Request body

```
{
  "Info": {
    "Title": "AIRR Data Commons API",
    "description": "API response for
        ↪ repertoire query",
    "version": 1.3,
    "contact": {
      "name": "AIRR Community",
      "url": "https://github.com/airr-
          ↪ community"
    }
  },
  "Rearrangement": [
    {
      "v_call": "TRBV5-6*01",
      "d_call": "",
      "j_call": "TRBJ1-3*01",
      "c_call": null,
      "junction": "
          ↪ TGTGCCAGCAGCTTACGCACTGCGGGAAACACCATATATTTT
          ↪ ",
      "junction_aa": "CASSLRTAGNTIYF",
      "duplicate_count": {},
      "cell_id": null
    },
    {
      "v_call": "TRBV7-3*01",
      "d_call": "",
      "j_call": "TRBJ2-2*01",
      "c_call": null,
      "junction": "
          ↪ TGTGCCAGCAGCTTGGCTGGATCGGGGGAGCTGTTTTTT
          ↪ ",
      "junction_aa": "CASSLAGSGELFF",
      "duplicate_count": {},
      "cell_id": null
    }
  ]
}
```

(b) Response

Figure 4.8: An example request to the endpoint `POST /v1/rearrangement`. Notice that the rearrangements returned don't have all their fields as specified by `"include_fields"` and that the number of repertoires returned is small as specified by `"from"` and `"size"`.

```
{
  "facets": "study.study_id",
  "filters": {
    "op": "in",
    "content": {
      "field": "repertoire_id",
      "value": [
        "395",
        "394",
        "393",
        "392",
        "391",
        "5e57da9c517fd62586a48529"
      ]
    }
  }
}
```

(a) Request body

```
{
  "Info": {
    "Title": "AIRR Data Commons API",
    "description": "API response for
        ↪ repertoire query",
    "version": 1.3,
    "contact": {
      "name": "AIRR Community",
      "url": "https://github.com/airr-
          ↪ community"
    }
  },
  "Facet": [
    {
      "study_id": "toy_study",
      "count": 1
    },
    {
      "study_id": "PRJNA248411",
      "count": 4
    }
  ]
}
```

(b) Response

Figure 4.9: An example facets request to the endpoint `POST /v1/repertoire`. Notice that the `"facets"` parameter interacts with the `"filters"` parameter to restrict the repertoires considered for counting, if it was omitted the counting would be performed over all of the repertoires in the repository.

```
{
  "name": "airr-api-ireceptor",
  "version": "0.1.0",
  "last_update": null
}
```

(a) `GET /v1/info`

```
{
  "result": "success"
}
```

(b) `GET /v1`

Figure 4.10: Example responses from the metadata endpoints.

the repertoire belongs to (ADCb, DCb). As mentioned a study can contain multiple repertoires, and so the `study` field is duplicated between all the repertoires that belong to that study.

- Rearrangement: represents information about DNA sequence. A lot of fields compose this resource but the most important are `sequence_id` which is the resource's ID and `repertoire_id` which is the repertoire to which the rearrangement belongs to (ADCa, DCa).

ADC protection

A mapping needs to be defined from the requirements for access levels and ADC models to UMA concepts:

- Studies: a study is the top-level resource of the API and a researcher is granted access to a study, so it maps easily to the UMA concept of resources in a one-to-one manner. A UMA resource has fields like `name` and `type` which can be used to add information about the study, though the UMA resource ID cannot be the same as the study ID since there is no way to set this ID in UMA 2, therefore a mapping needs to be maintained somewhere between the UMA resource ID and study ID, as mentioned previously. This UMA resource ID is important because it is used for all of the interactions with the UMA authorization server such as when emitting the permissions ticket and it is returned when introspecting RPT tokens.

- Access levels: 2 access levels are defined which can be thought of as not intersecting in their permissions. The mapping is simple where the scopes might be one of `statistics`, `raw_sequence` where an access level might have more than one associated scope. Access level 1 would correspond to the `statistics` scope and level 2 to the `statistics`, `raw_sequence` scopes, tough this translation is purely conceptual, and the implementation does not make use the concept of access levels but of the scopes to which they map.

  The ADC's repertoire resource has fields that belong to both access levels, even public information. The rearrangement resource's fields only belong to access level 2.

Protecting the current ADC API v1 with the requirements presented above has several issues:

- The mapping between the study IDs and the UMA resource IDs. Ideally, the mapping should be maintained in the middleware server, but this introduces the problem of state synchronization between the authorization server (Keycloak), middleware server and the repository, given that as studies are created and deleted in the repository (by adding, updating and deleting repertoires). This information needs to propagate to the authorization server and the mappings between the UMA resources and studies must be updated in the middleware server, either directly from the repository (which means code change) or indirectly by the party making the modifications. It is assumed that the modifications to the repository databases are made only by the repository admins.

- Some ADC resources have information belonging to multiple access levels, even public information. For example the information about the studies and the repertoire's ID, such as the fields `repertoire_id`, `study.study_id`, `study.study_title`, is considered not sensitive and is intended to be publicly accessible. Other information like the subject's personal information, such as the `subject` field, can be classified as access level 1 (`statistics` scope) and some other information relating to DNA sequences can be classified as level 2, such as the `sample` field (`raw_sequence` scope). The final classification of fields into their access levels is still being worked on by the partners of the project. Currently, all of the ADC's rearrangement fields are classified as level 2 (`raw_sequence` scope).

  This introduces the problem of how to expose only the information that the user has access to, such as a user having access to a study only to level 1 and attempting to access the individual ADC endpoint for the repertoire, how should the `sample` field which is also returned be handled. Additionally, how should a user that has not been granted access to any resource be able to access the public information related to repertoires?

- The translation between a repertoire's or rearrangement's ID and the UMA resource ID to which it belongs from the information present in the request or its corresponding response from the repository. For repertoires there is the `study.study_id` field which when used in conjunction with the mapping from study ID to UMA resource ID can be used to associate a repertoire with a UMA resource. For rearrangements there is the `repertoire_id` field which when used in conjunction with the repertoire's `study.study_id` field and the mapping to UMA resource ID can be used to associate a rearrangement to a UMA resource. Given that in the request for the individual endpoints the study ID is not specified in the URL, that for the search endpoints by default the resource IDs that are requested are not present in the body of the request, that the user can choose specific fields to be returned by a search which might involve omitting the resource IDs from the repository's response and that the search endpoints' facets function does not return the resource ID's that were used in the computation at all there needs to be an approach to obtain these resource IDs from the requests or responses or to enforce that only the resources present in the RPT token are considered by the repository.

- Request duplication, or worse, for most ADC requests. Using the existing API's responses to obtain the IDs has the disadvantage of duplicating most requests to the repository, once for getting the IDs to emit a permissions ticket and again to obtain the resources given an RPT token. Additionally in a simplistic approach, a single request might involve multiple requests, for example when requesting a rearrangement internally an additional request would be made to a repertoire to obtain the study ID for the mapping to the UMA resource ID (the rearrangement contains the repertoire ID, and the repertoire contains the study ID, while the middleware contains the mappings between the study ID and resource UMA ID). This can be somewhat mitigated with a cache as discussed below.

Two approaches were taken to tackle these problems: by modifying the ADC API to be more UMA aware and present the information about the resource's study ID in the user's request and by engineering a solution that tries to mitigate some of these problems, using as mentioned a cache for resource IDs and by modifying the user's requests. Modifying the ADC API might lead to the simplest middleware solution but using the responses from the endpoints has the advantage of reusing an existing API specification and existing implementations of that API.

4.4.2.1 ADC API extensions

As mentioned, the current ADC API is not optimized to allow requests to be filtered by a middleware server using UMA 2. Given the inadequacies previously mentioned an improvement is proposed that simplifies the API and allows the requests to be validated and filtered with greater efficiency. The main changes can be described in three principles:

- Change the endpoints that perform a search and return a list of objects or that return the full list of objects (`POST /v1/repertoire` and `POST /v1/rearrangement`) and have them return instead of a list of ID pairs, the study ID and the resource ID of the corresponding resource, that can be returned in URL form containing both IDs. A client of the API would use these endpoints to perform a search query, and then for each of the returned ID pairs would access each resource individually.
- Add the study ID as part of the URL path and specify that any implementation of the improved API must reject a resource request in which the resource ID specified in the path is not under the study ID specified in the path.
- Any resource returned by the API must belong to only one access level. The repertoire resource response specification needs to remove fields from access level 2 (for example the `sample` field), and extract them to a separate resource or move them to another resource. It is important that a request approved by the middleware server for a specific access level not return information from an access level the user does not have access to.

  Additionally, some ADC implementing repositories have additional non-compliant fields for their resources and this can lead to access level leak, when an endpoint for an individual resource that should only return information for a specific access level also returns information from a higher access level, although this would be considered an implementation bug in the proposed API extension.

An API that obeys these principles would have two main classes of endpoints:

- Public or minimally restricted endpoints that perform a search or listing of all resources that return resource and study IDs, which might be returned in URL form, potentially containing additional IDs related to resources under studies but above the resource ID.

  The search endpoints should be protected if the search is done over protected fields since the presence of a resource matching a query can be used to leak information. If the search is done over public fields the endpoint can be public.

ADC v1 Endpoints	New Proposed Endpoints
GET /v1/repertoire/{id}	(1) GET /v2/study/{id}/repertoire/{id}
POST /v1/repertoire	(2) GET /v2/search/repertoire
GET /v1/rearrangement/{id}	(3) GET /v2/study/{id}/repertoire/{id}/rearrangement/{id}
POST /v1/rearrangement	(4) GET /v2/search/rearrangement

Table 4.2: ADC API to extensions endpoints mapping.

- Individual resource endpoints that given the study ID and resource ID return the resource. These individual resources would be protected according to the researcher's study and access level according to the flow described above.

  The middleware server would filter the researcher's requests and either grant or deny access to a resource according to the RPT token presented. Upon introspection, the RPT token indicates the UMA IDs to which the researcher has access and their access level. The middleware would compare the UMA ID to the study ID in the URL path which the researcher is trying to access, using the mapping from UMA ID to study ID mentioned previously, and if they match and allow access, the access is permitted and forwarded unmodified.

  Accessing resources can potentially have lower performance because for a search each returned resource needs to be individually accessed, implying an HTTP request for each resource found instead of being returned the full object list on the search endpoint. If needed, this can be mitigated by creating endpoints that allow a user to specify the list of resources to access, for example, the list of resource URLs, returning the requested list, or a subset according to the access given by the RPT token. This request would be parseable by the middleware server for the purposes of determining which resources and corresponding study are being accessed. So a researcher would perform a search to obtain the list of resources and then use this endpoint, using the flow described above, to get the ticket and then the RPT token to access this list of resources or a subset.

For the ADC API, the main proposed changes can be seen in table 4.2. In table 4.3 you can see a more in-depth description of each new endpoint's behaviour. Endpoints `/v1, /v1/info` are not relevant for the proposed authorization design and can remain unchanged and public. The proposed changes in terms of semantic versioning would be considered breaking changes, hence the v2. For endpoints (1) and (3) the resource server implementing the API must validate that the resource IDs belong to the study or repertoire in the path to avoid leaks. The response for the endpoint (1) should not return information related to DNA sequences (access level 2). Endpoint (5) is optional given that the same effect can be done accessing the individual endpoints for each resource, but can be needed given that individual access for each resource plus the UMA overhead can decrease performance significantly, this endpoint can be fully implemented in the reverse-proxy middleman server. Any additional resources or endpoints that are to be added should follow the same 3 principles mentioned above.

This approach to the problem has some advantages, namely:

Endpoints	**Description**	**Access Level**
(1) GET /v2/study/{id}/repertoire/{id}	Returns individual repertoire that belongs to the specified study in the path.	1
(2) GET /v2/search/repertoire	Returns list of repertoire and study IDs pairs matching user search.	public, 1
(3) GET /v2/study/{id}/repertoire/{id} /rearrangement/{id}	Returns individual rearrangement that belongs to the specified study and repertoire in the path.	2
(4) GET /v2/search/rearrangement	Returns list of rearrangement, repertoire and study IDs triplets matching user search.	2
(5) GET /v2/resources	Given a list of resource ID tuples (in the form of an URL) returns the matching resources that the user is given access to according to the RPT token. Can return any type of resource such as repertoires, rearrangements, etc.	1,2

Table 4.3: ADC API extensions endpoint descriptions.

- There is no need to modify the user request (for example the `"fields"` or `"filters"` ADC query parameters) since all of the information that the middleware server needs to decide whether to approve the request is present in the request URL or body.

- A simpler middleware server implementation, since there is no need to process the returned information, or to keep a mapping of IDs (besides the study ID to UMA resource ID) in the server, which is used to obtain the UMA resource ID from a repertoire's or rearrangement's ID.

While this approach has the advantages mentioned above it is only really useful when there is broad support from the repository maintainers willing to implement the new API. This could be helped by making this API extension an official AIRR standard, that is, ADC v2. This approach was pitched to the maintainers of the iReceptor Turnkey project, a repository implementing the current ADC API, and to representatives of the AIRR community knowledgeable in the ADC API. A draft proposal of the new API extensions was sent, which you can find in Appendix A. While the members were supportive of the middleware server approach to the problem of access control, some concerns were raised, mainly:

- That the ADC resources such as repertoires and rearrangements don't have a direct hierarchy in the official API specification, meaning, making this hierarchy rigid in the API would likely be rejected by the users. Although in practice such a hierarchy exists.

- That the API extensions would involve making a lot of changes that would require additional work from repository maintainers which might prefer to keep using the previous API version.

- That the approval of the API extensions into the official ADC API would take a lot of time and might suffer changes in the meantime which would dilute the original intent of this proposal.

Besides the problems identified as part of the pitch of API extensions, several additional limitations were identified:

- The access level (including public information) that corresponds to each resource field is still not finalized and might suffer changes in the future. The API extensions codify in the API what field is which access level (to avoid access level leaks) and a more flexible approach might be desirable.

- The listing endpoints (endpoint 5 in table 4.3) accept a list of resources to be returned by the repository. In the case of rearrangements, this list can be very large, and lead to performance limitations which this approach is trying to avoid.

- The search endpoints still need to be protected somehow when the search is done over protected resource fields in order not to leak information. This can be done by using a UMA resource for each search endpoint but this still has some issues.

Given these limitations this approach to the problem was rejected, instead opting for protecting the existing ADC API as described in the next section. Though this approach is still useful to consider when designing a new API, from scratch, that needs to be protected.

4.4.2.2 ADC API v1

Given the limitations of the ADC extensions approach, this is the approach chosen to solve the problem, which involves protecting the existing, current ADC API. This approach involves modifying some user's ADC requests in the case of the search (and facets) endpoints and response field filtering of resources based on the user's access level.

The solution's reference implementation, built alongside this thesis, is built using Java 11 on Java Spring and is open source (Rosa, osa). The implementation is provided as a Docker image, to facilitate integration with existing ADC repositories. Additionally, an example docker-compose configuration file is provided to allow the user to more easily experiment with the middleware. The features described here are tested with unit tests that run automatically in a CI server. The docker images are uploaded automatically to Dockerhub when a commit is tagged with a semantic version.

The solution's components and flow can be described in figure 4.11. The components that compose the solution can be seen, such as the middleware's ID Cache which contains an SQL database for permanent storage and a volatile cache such as Redis for performance-related optimizations. The steps described in the flow are the same as the steps in the diagram 4.4, the only differences being in:

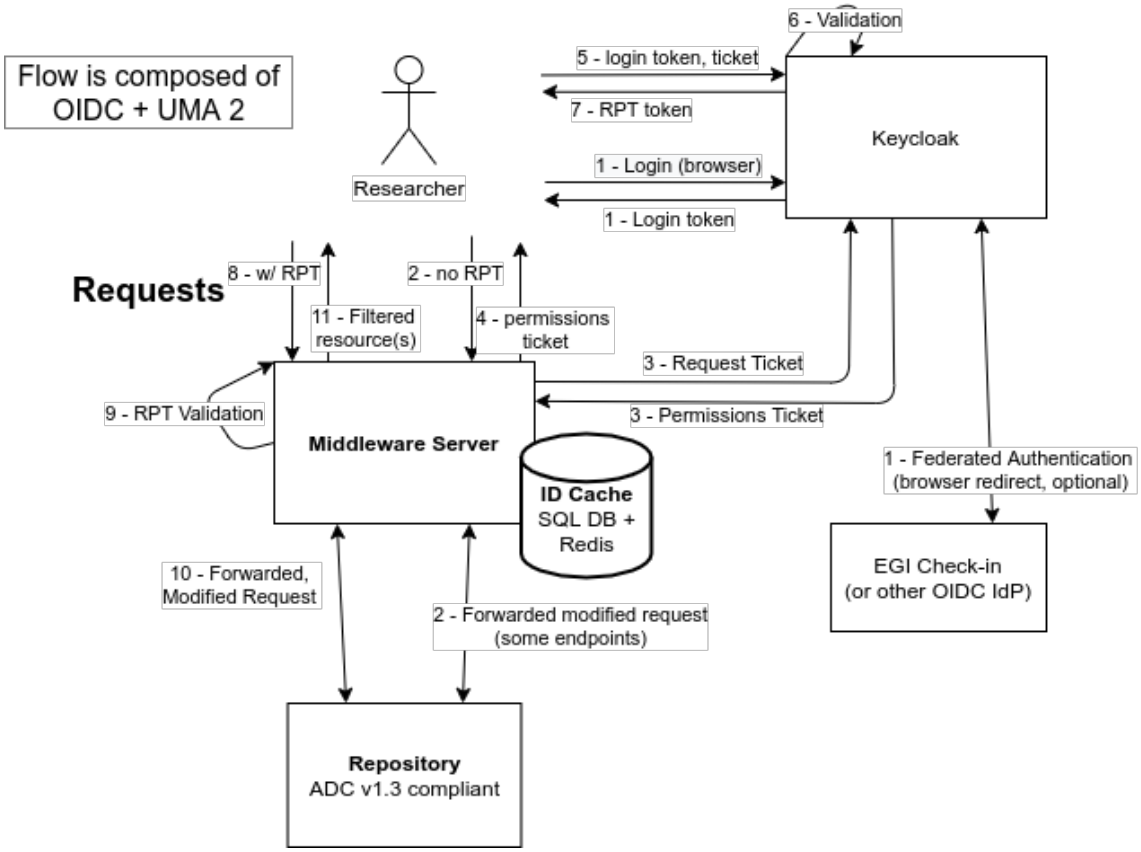


Figure 4.11: Researcher's flow for accessing resources using the middleware server and other relevant components.

- Step 1: The full flow for the researcher's login is shown, using only Keycloak or using a third party OIDC IdP such as ORCID or EGI Check-in.
- Step 2: Request Forwarding. Some ADC endpoints protected by this approach must be called twice, namely `GET /v1/rearrangement/{id}` (sometimes), `POST /v1/repertoire` and `POST /v1/rearrangement`.

  The middleware forwards the modified user's HTTP request to the repository to obtain the resource IDs the original request would have accessed, then these are mapped to their corresponding UMA resource IDs and used to emit a permissions ticket.

  The endpoint `GET /v1/repertoire/{id}` does not make use of this forwarding.

Resource Fields Access Level & CSV configuration file

As mentioned before, the ADC repertoire's and rearrangement's fields have a corresponding access level assigned to them. For public fields, there is no associated access level and no associated UMA scopes (referenced as access level 0 in this section), for access level 1 the `statistics` UMA scope is associated and for level 2 the `raw_sequence` scope (though a user with access level 2 would also have access to fields with access level 1).

Endpoint	Protection	Description
(A) POST /v1/synchronize	password	Synchronize the middleware's ID cache and update Keycloak's UMA resources.
(B) GET /v1/public_fields	public	Returns the public fields available for each resource: Repertoire and Rearrangement. Useful for the search endpoints.

Table 4.4: Added endpoints specific for the middleware. (A) is password protected which is sent as a Bearer value in the Authorization header and (B) is publicly accessible.

In the reference implementation only the fields `"repertoire_id"`, `"study.study_id"` and `"study.study_title"` for the repertoire and fields `"sequence_id"` and `"repertoire_id"` for rearrangements are mandatory to be present and have an associated access level. Any other fields can be configured, with any name and any access level and these fields can represent nested objects and lists of objects (in which case all objects in the list must have the field part). While 3 access levels are defined (including 0 for public), the implementation supports an arbitrary set of access levels (as represented by a scope). Configuration of the fields and their protection can be done with a CSV file, called the CSV configuration file from now on.

For an example configuration see figure 4.12. Notice that some fields (column `field`) are marked as public (see column `protection`), and some are marked as protected with scopes `statistics`, `raw_sequence`, `scope_2` and `arbitrary_scope` (column `access_scope`) where the scopes are completely arbitrary and user defined. The field names besides the mandatory ones are also arbitrary and user defined. The `class` column specifies whether the field is for a `Repertoire` or `Rearrangement` resource (only these 2 values are valid). Column `protection` can only have values `public` or `protected`. Column `field_type` specifies the field value type, for user query validation, and can be one of `string`, `boolean`, `number`, `integer` or `array_string`. Column `include_fields` can be one of `miairr`, `airr-core` or `airr-schema` and specifies what field belongs to each of the ADC query's `"include_fields"` parameter value. Column `description` is unused. For the current file in use see Appendix B.

Synchronization & ID Cache

```
class,field,protection,access_scope,field_type,include_fields,description
Repertoire,repertoire_id,public,,string,airr-core,Identifier for the ...
Repertoire,study.study_id,public,,string,miairr,Unique ID ...
Repertoire,study.study_title,protected,statistics,string,miairr,Descriptive study title
Repertoire,field1.other_field.some_other,protected,scope_2,boolean,miairr,Some field
Repertoire,sample.tissue,protected,raw_sequence,string,miairr,The actual tissue ...
Rearrangement,sequence_id,protected,raw_sequence,string,airr-core,Unique query ...
Rearrangement,arbitrary_field,protected,arbitrary_scope,number,airr-core,The query ...
Rearrangement,repertoire_id,protected,raw_sequence,string,airr-schema,Identifier to ...
```

Figure 4.12: An example CSV configuration file that can be used by the reference implementation.

As mentioned before, this approach has the limitation of adding additional HTTP requests for each endpoint which can be somewhat mitigated by an ID cache.

The ID cache is composed of an SQL database (permanent storage), which is used to store both the mappings between the UMA resource IDs and the study IDs that correspond to it (two-way mapping) and the mappings between repertoire IDs and the study IDs to which they belong (one-way mapping) allowing to get the UMA resource ID given a repertoire ID.

The ID cache is additionally composed of a volatile database (such as a Redis cache) that stores rearrangement IDs and the repertoire IDs to which they belong (one-way mapping from the rearrangement to the repertoire), though in the case of Redis since the user controls its configuration the amount of volatility is user-configurable such as the maximum amount of rearrangement mappings stored or whether the cache is restored between restarts. The rearrangement IDs are not stored permanently, even though they could be, because of the number of rearrangements that a repository can contain (many thousands) which significantly increases the size of the database and the time needed to synchronize the ID cache with the repository. Additionally, the volatile database can store the mappings between studies and repertoires, the same as the SQL database, which is done because the Redis cache is usually faster than accessing the same value in the SQL database, especially if the SQL server is in another server and Redis is in the same server as the middleware. Though this case is not considered in the descriptions or pseudo-code to keep it more simple.

In the reference implementation, the database component can be any SQL database supported by Java Spring such as h2, MySQL, or Postgres. The cache component can be either Java Spring's internal cache or Redis or even disabled. Which one is used is configurable in a Java properties file, which includes additional middleware configuration fields such as the password hash for the synchronization endpoint.

The middleware needs to synchronize its ID cache with the underlying repository to update the ID cache and to update any UMA resources to match the studies in the repository, which is accomplished with the help of the synchronization endpoint (see A in table 4.4). Calling this endpoint will trigger the synchronization flow in the middleware and either success is returned on successful synchronization or an error is returned when synchronization fails, which can happen when updating the cache fails or updating or creating a UMA resource fails. The endpoint is password-protected, where a hash of the password is stored in the middleware's configuration file, see figure 4.13 for an example. The synchronization flow using this endpoint has the following steps:

1. The user (repository admin) calls the synchronization endpoint with the password for this endpoint. On an incorrect password, an error is returned and no synchronization is performed. The middleware will check the hash of the request password against the hash supplied in the configuration file using the BCrypt password hasher with 10 rounds. The user should ideally call this endpoint right after the first setup of the middleware or right after any data in the protected repository is modified (repertoires or rearrangements are added, updated, or deleted).

2. The middleware deletes the cache for the rearrangement IDs and the database for the repertoire IDs.

3. The middleware requests the full list of repertoires in the repository using the `POST /v1/repertoire` endpoint, see figure 4.14 for an example (repertoires include fields like `"repertoire_id"`, `"study.study_id"` and `"study.study_title"`). From these the set of studies is constructed, using the study ID field along with additional information like the title (when multiple repertoires belong to the same study they must return the same `"study"` field). Additionally, the set of UMA resources in Keycloak is requested (if none exist the set is empty) using the list resource descriptions endpoint from the UMA 2 resource registration API (which Keycloak supports), see figure 4.15 for an example. These 2 sets are compared with the mappings in the database (if there are none the set of mappings is empty) in the following order:

   (a) If there are any mapping entries in the database that don't map to a corresponding repository study they are deleted from the database.

   (b) If there are any UMA resources in Keycloak that don't have a corresponding entry in the mappings database they are deleted from Keycloak, using UMA's delete resource registration endpoint (see figure 4.19 for an example).

   (c) If there are any mapping entries in the database that don't map to a corresponding UMA resource they are deleted from the database.

   (d) If there are studies from the repository that don't have a corresponding mapping in the database they are created in Keycloak and a mapping entry is added. The endpoint for creating a UMA resource is UMA's create resource description endpoint (see figure 4.16 for an example) and it returns the UMA resource ID on success which is stored in the database alongside its corresponding study ID. The created UMA resource contains the study title from the repertoire and the scopes specified in the field configuration file (see below). Additionally, the resource owner's username is specified on resource creation and the corresponding user is set as the resource owner for that resource, since Keycloak does not, as of the time of writing, allow changing resource owners after creation. The username that is used is taken from the configuration file and is the same for all of the created resources.

   (e) If there are any intersecting resources between the Keycloak resources and the database mappings they are validated. This involves requesting the resource from Keycloak, using UMA's read resource description endpoint (see figure 4.17), and updating it, using UMA's update resource description endpoint (see figure 4.18) if there are any missing scopes when compared to the scopes from the CSV configuration file.

4. The middleware using the repertoire list from the previous step updates the repertoire ID to study ID mappings. Given that this mapping was deleted in a previous step this involves simply creating a mapping for each repertoire returned.

An example of Keycloak's list resource descriptions endpoint can be seen in figure 4.15. In the example the endpoint `GET /auth/realms/master/authz/protection/resource_set` is used to obtain the full list of UMA resource IDs. The base path of this endpoint is specified by Keycloak's UMA discovery document for the specific realm.

An example of Keycloak's create resource description endpoint can be seen in figure 4.16. In the example the endpoint `POST /auth/realms/master/authz/protection/resource_set` is used create a UMA resource and obtain the UMA ID. Notice that the request body has some additional fields than those specified in the specification: the `"ownerManagedAccess"` and `"owner"` which in this example sets the user `"rosald"` as the resource owner for the created resource. Also, notice that the study ID and title are used to set the UMA resource name.

An example of Keycloak's read resource description endpoint can be seen figure 4.17. In the example the endpoint `GET /auth/realms/master/authz/protection/resource_set/dbed4e3e-07e4-4750-b834-805f950a9b88` is used to obtain the UMA resource description. Notice that the JSON schema for the `"resource_scopes"` field is incompatible with the specification since it is supposed to return a list of JSON strings and not objects.

An example of Keycloak's update resource description endpoint can be seen figure 4.18. In the example the endpoint `PUT /auth/realms/master/authz/protection/resource_set/dbed4e3e-07e4-4750-b834-805f950a9b88` is used to update the UMA resource scopes. The scope `"some_other_scope"` was the resource's scope before the request and the request adds the scopes `"raw_sequence"` and `"statistics"`. Notice the presence of the `"name"` field which must be the same as the resource name (a Keycloak limitation). Keycloak does not return a body for this request while the specification mandates that the resource ID must be returned, although the body is discarded by the middleware in either case.

The synchronization flow can be summarized with the following pythonic pseudo-code:

```
def getAllScopes():
  fieldsCsv = csv.loadFile(config.load("app.adcCsvConfigPath"))
  scopes = fieldsCsv.select("access_scope")
  return set(scopes)

# POST /v1/synchronize
def synchronize(password):
  # step 1, check password
  syncPassHash = config.load("app.synchronizePasswordHash")
  if BCrypt.hash(password, 10) != syncPassHash:
    raise "Invalid password"

  repository = AdcClient("http://repository:80/airr/v1")
  uma = UmaClient("http://Keycloak:8082")
  db = DbRepository("http://postgres:5432")
  cache = Cache("http://redis:6379")
  allUmaScopes = getAllScopes()

  # step 2, delete cache
  cache.deleteAll()
  db.deleteStudyRepertoireMappings()
```

```
# step 3, synchronize studies
body = {
  "fields": [
    "repertoire_id", "study.study_id", "study.study_title"
  ]
}
repertoires = repository.getRepertoires(body)
repositoryStudies = {
  rep["study"]["study_id"]: rep["study"]["study_title"] for rep in repertoires
}
repositoryStudyIds = set(repositoryStudies.keys())
umaIds = set(uma.listUmaResourceIds())

# 3. (a), delete DB studies not in repository
for studyId in set(db.studyIds()) - repositoryStudyIds:
  db.deleteUmaStudyMapping(studyId)

# 3. (b), delete UMA resources not in DB
for umaId in umaIds - set(db.umaIds()):
  uma.deleteResource(umaId)

# 3. (c), delete DB studies not in UMA server
for umaId in set(db.umaIds()) - umaIds:
  studyId = db.getStudy(umaId)
  db.deleteUmaStudyMapping(studyId)

# 3. (d), create UMA and DB studies not in repository
for studyId in repositoryStudyIds - set(db.studyIds()):
  umaResource = {
    "name": "study ID: " + studyId + "; title: " + repositoryStudies[studyId],
    "type": "study",
    "resource_scopes": allUmaScopes,
    "ownerManagedAccess" : True,
    "owner": config.load("uma.resourceOwner")
  }

  createdUmaId = uma.createResource(umaResource)["_id"]
  db.addUmaStudyMapping(createdUmaId, studyId)

# 3. (e), validate UMA studies in repository
for umaId in umaIds - set(db.umaIds()):
  umaResource = uma.getResource(umaId)
  actualScopes = set([
    e["name"] for e in umaResource["resource_scopes"]
  ])

  if not actualScopes.contains(allUmaScopes):
    updateUmaResource = {
      "name": umaResource["name"],
      "resource_scopes": actualScopes + allUmaScopes # no duplicates
    }
    uma.updateResource(umaId, updateUmaResource)

# step 4, synchronize repertoires
for repertorie in repertoires:
  db.addStudyRepertoireMapping(
```

```
curl --location --request POST \
  'localhost:8080/airr/v1/synchronize' \
  --header 'Authorization: Bearer master'
```

(a) Example curl request to the synchronize endpoint with password: 'master'.

```
// on success
{
    "status": 200
}

// on invalid password or
// connection error to Keycloak or DB
{
    "status": 401
}
```

(b) Example Responses.

Figure 4.13: An example request to the endpoint `POST /v1/synchronize` with the synchronization password and some example responses.

```
    repertoire["study"]["study_id"],
    repertoire["repertoire_id"])
```

ADC API Responses

For the individual endpoints and the search endpoints (excluding facets), access level based field filtering is performed. This filtering is performed when a user supplies an RPT token with his request and the repository returns resources for that request.

The middleware takes from the RPT token, using UMA's introspection endpoint, the resources and the scopes to which the user has access to, then these scopes are translated into a list of valid fields (for the ADC resource type considered) for each UMA ID present in the token, including additionally the list of public fields for the resources. These public fields and the scope associated with each field is specified in the CSV configuration file.

The middleware forwards the request to the repository and performs filtering based on the resources returned: for each resource, the fields are matched against the valid fields from the previous step, and if any don't match, they aren't returned by the middleware. Field filtering can handle nested objects and nested lists of objects (demarked by a dot in the CSV configuration file's field names, for example, `study.study_id`). If in a list of resources no scopes or public fields for a resource are granted the resource is filtered out.

See figure 4.20 for an example filtered repertoire. In the figure, compared to the original, full repertoire in figure 4.5, which can be obtained if the request RPT token grants access to all of the scopes corresponding to the repertoire, the example in (a) is missing the `"sample"` field since the response was obtained with an RPT token only granting access to `statistics` scope while the `"sample"` field is configured as needing `"raw_sequence"` scope to access. In example (b) only the public fields for the repertoires are displayed, which can be obtained by using the search endpoint while requesting only these public fields, or while requesting the individual repertoire with an RPT token that has no scopes associated for the repertoire. The rearrangements by the default configuration have no public fields and contain only one scope, so in practice, no field filtering on them is performed (except to filter out whole resources).

Additionally, to improve memory usage and request latency, the middleware will perform this filtering in streaming mode, which has the disadvantage that if malformed JSON is returned by

```
{
    "fields": [
        "study.study_title",
        "study.study_id",
        "repertoire_id"
    ]
}
```

(a) Request body

```
{
  "Info": {
    // ...
  },
  "Repertoire": [
    {
      "study": {
        "study_title": "The Different T-cell
            ↪  ...",
        "study_id": "PRJNA330606"
      },
      "repertoire_id": "5
          ↪ e53de7f9463684866be6092"
    },
    {
      "study": {
        "study_title": "toy study",
        "study_id": "toy_study"
      },
      "repertoire_id": "5
          ↪ e68d4c206e94e498f47a732"
    }
  ]
}
```

(b) Example response

Figure 4.14: An example request made by the middleware to the repository on the endpoint `POST /v1/repertoire` to obtain the full list of repertoires.

```
[
    "dbed4e3e-07e4-4750-b834-805f950a9b88",
    "2fb65e02-52c8-4ab1-8219-aa07aaf35f64",
    "058a9555-18d7-4e71-9299-f59c3043a42e"
]
```

Figure 4.15: An example response to the request made by the middleware to Keycloak on the list resource descriptions endpoint.

```
{
  "name": "study ID: toy_study; title: toy
      ↪ study",
  "type": "study",
  "ownerManagedAccess": true,
  "resource_scopes": [
    "raw_sequence",
    "statistics"
  ],
  "owner": "rosald"
}
```

(a) Request body

```
{
   "_id":"058a9555-18d7-4e71-9299-
       ↪ f59c3043a42e"
}
```

(b) Example response

Figure 4.16: An example request made by the middleware to Keycloak on the create resource description endpoint.

```
{
  "name": "study ID: PRJNA248411; title: Immunoglobulin class-switched B cells provide
      ↪ an active immune axis between CNS and periphery in multiple sclerosis",
  "type": "study",
  "owner": {
    "id": "a745713b-fed9-4b44-95d6-9c8612c02286"
  },
  "ownerManagedAccess": true,
  "attributes": {},
  "_id": "dbed4e3e-07e4-4750-b834-805f950a9b88",
  "uris": [],
  "resource_scopes": [
    {
      "name": "raw_sequence"
    },
    {
      "name": "statistics"
    }
  ],
  "scopes": [
    {
      "name": "raw_sequence"
    },
    {
      "name": "statistics"
    }
  ]
}
```


Figure 4.17: An example response to the request made by the middleware to Keycloak on the read resource description endpoint.


```
{
  "name": "study ID: PRJNA248411; title: Immunoglobulin class-switched B cells provide
      ↪ an active immune axis between CNS and periphery in multiple sclerosis",
  "type": "study",
  "resource_scopes": [
    "some_other_scope",
    "raw_sequence",
    "statistics"
  ]
}
```


Figure 4.18: An example response to the request made by the middleware to Keycloak on the update resource description endpoint.

```
// request
DELETE /rreg/dbed4e3e-07e4-4750-b834-805f950a9b88
Authorization: Bearer 23123adca

// response
204 No content
```

Figure 4.19: An example request and response middleware to Keycloak to delete an UMA resource. `rreg` is Keycloak's base path for the resource registration API which can be found in Keycloak's UMA discovery document.

```
{
  "Info": {
    "Title": "AIRR Data Commons API",
    "description": "API response for
        ↪ repertoire query",
    "version": 1.3,
    "contact": {
      "name": "AIRR Community",
      "url": "https://github.com/airr-
          ↪ community"
    }
  },
  "Repertoire": [
    {
      "repertoire_id": "5
          ↪ e53de7f9463684866be6092",
      "repertoire_name": null,
      "repertoire_description": null,
      "study": {
        "study_id": "PRJNA330606",
        "study_title": "The Different T-cell
            ↪  ...",
        "other fields": "..." // other
            ↪ fields
      },
      "subject": {
        "subject_id": "BC0001A",
        "synthetic": false,
        "other fields": "..." // other
            ↪ fields
      },
      "data_processing": [
        {
          "data_processing_id": "5
              ↪ e53de7f9463684866be6092",
          "primary_annotation": true,
          "other fields": "..." // other
              ↪ fields
        }
      ]
    }
  ]
}
```


(a) Partially filtered repertoire


```
{
  "Info": {
    "Title": "AIRR Data Commons API",
    "description": "API response for
        ↪ repertoire query",
    "version": 1.3,
    "contact": {
      "name": "AIRR Community",
      "url": "https://github.com/airr-
          ↪ community"
    }
  },
  "Repertoire": [
    {
      "repertoire_id": "5
          ↪ e53de7f9463684866be6092",
      "repertoire_name": null,
      "repertoire_description": null,
      "study": {
        "study_id": "PRJNA330606",
        "study_title": "The Different T-
            ↪ cell ...",
      }
    }
  ]
}
```


(b) Repertoire public information

Figure 4.20: Examples of a filtered repertoire based on the CSV configuration file found in appendix B.

the repository in the middle of the response the middleware will still return a 200 status code, but also with malformed JSON.

This filtering algorithm can be described in more detail in the following pythonic pseudo-code:

```
fieldsCsv = csv.loadFile(config.load("app.adcCsvConfigPath"))

def getClassProtectedFields(className, scopes):
  fields = fieldsCsv.select("field")
              .whereEquals("class", className)
              .whereEquals("protection", "protected")
              .whereIncluded("access_scope", scopes)
  return set(fields)

def getClassPublicFields(className, scopes):
```

```
  fields = fieldsCsv.select("field")
              .whereEquals("class", className)
              .whereEquals("protection", "public")
  return set(fields)

def umaResourcesToGrantedFields(umaResources, className):
  grantsUmaScopeMap = {
    resource["resource_id"]: set(resource["resource_scopes"])
    for resource in umaResources
  }

  umaGrantedFieldsMap = {
    umaId: getClassProtectedFields(className, scopes) +
              getClassPublicFields(className)
    for umaId, scopes in enumerate(grantsUmaScopeMap)
  }
  # access only public fields when access is denied
  umaGrantedFieldsMap[None] = getClassPublicFields(className)

  return umaGrantedFieldsMap

def filterResourceRecursive(resource, scopes):
  splitScopes = [
    (s.split(".", 1)[0], s.split(".", 1)[1]) for s in scopes
  ]
  collapsedScopes = {
    topField: set()
    for (topField, rest) in splitScopes
  }
  for (topField, rest) in splitScopes:
    collapsedScopes[topField].add(rest)

  for k, v in enumerate(resource):
    if not k in collapsedScopes:
      del resource[k]
      continue

    innerFields = collapsedScopes[k]
    if innerFields.empty():
      continue

    if type(v) is list:
      for nested in v:
        filterResourceRecursive(nested, innerFields)
      for i, nested in v:
        if nested.empty():
          del resource[k][i]

    else if type(v) is dict:
      filterResourceRecursive(v, innerFields)
      if v.empty():
        del resource[k]

    else:
      log.error("bad CSV config")
      del resource[k]
```

```
def streamFilteredResources(adcResponseStream, umaGrantedFieldsMap, responseResourcesName,
↪  resourceUmaIdGetter):
  responseStream = createResponseJsonStream()
  responseStream.startObject()

  for field, valueStream in adcResponseStream.streamObject():
    if field == "Info":
      responseStream.sendField("Info", valueStream.read())

    if field == responseResourcesName:
      responseStream.startArray(responseResourcesName)

      for resource in valueStream.streamList():
        umaId = resourceUmaIdGetter(resource)
        validFields = umaGrantedFieldsMap[umaId] if umaId in umaGrantedFieldsMap else
        ↪  umaGrantedFieldsMap[None]
        filterResourceRecursive(resource, validFields)
        if not resource.empty():
          responseStream.sendElement(resource)

      responseStream.endArray()

  responseStream.endObject()
  responseStream.close()
```

For the facets search functionality no filtering is performed, instead, relying on modifying the user query to restrict the resources considered for counting.

ADC API protection

To enforce access control, the middleware server validates the researcher's ADC requests, which might involve modifying them, and then forwards them to the repository. As mentioned above, using the full flow, some requests are made twice to the repository from the middleware.

In this subsection references are made to UMA's permission endpoint (see 2.8 for an example) and to UMA's token introspection endpoint (see 2.7 for an example). Keycloak implements both, and their base path is provided in the UMA discovery document Keycloak supplies.

For the individual ADC endpoints:

- `GET /v1/repertoire/{id}`: When no RPT token is supplied with the request the middleware matches the URL's requested ID against the repertoire IDs in the database, obtaining as well the associated UMA ID, and if the ID exists it emits a permissions ticket for that repertoire, by making use of UMA's permission endpoint, otherwise it returns HTTP's 404 error. In the reference implementation, the scopes requested for the ticket are obtained by parsing the CSV configuration file and obtaining the unique list of scopes that are present for the Repertoire class.

  When an RPT token is supplied it is introspected, using UMA's token introspection endpoint, to check which permissions were granted for the resource. The request is then forwarded to the middleware and the response's resource fields are filtered according to the scopes present in the RPT token (see above). Only one request in this flow is made to the repository to obtain a resource.

- `GET /v1/rearrangement/{id}`: It works the same as the above but the rearrangement ID supplied in the URL is checked against the volatile cache to obtain the associated repertoire ID and then the above flow is followed.

  If the cache does not have a mapping for the rearrangement's ID the middleware requests the rearrangement from the repository to obtain the repertoire ID which is in the cache. In this case, 2 requests are made to the repository in the flow to obtain a single rearrangement resource (and even 3 if the cache is wiped between the 2 requests without and with an RPT token).

The individual endpoints workings can be described in more detail with the following pythonic pseudo-code:

```
from filtering import umaResourcesToGrantedFields, streamFilteredResources

repository = AdcClient("http://repository:80/airr/v1")
uma = UmaClient("http://Keycloak:8082")
db = DbRepository("http://postgres:5432")
cache = Cache("http://redis:6379")

def getClassScopes(className):
  fieldsCsv = csv.loadFile(config.load("app.adcCsvConfigPath"))
  scopes = fieldsCsv.select("access_scope")
              .whereEquals("class", className)
  return set(scopes)

def validateToken(rptToken, umaIdsProducer, scopes):
  if rptToken == None:
    resources = [
      {
        "resource_id": umaId,
        "resource_scopes": scopes
      } for umaId in umaIdsProducer()
    ]

    ticket = uma.requestPermissionTicket(resources)['ticket']
    raise (401, { # emits permission ticket to the user
      'WWW-Authenticate:': 'UMA as_uri="http://Keycloak:8082", ticket="' + ticket + '"'
    })

  introspect = uma.introspectToken(rptToken)
  if not introspect['active']:
    raise 401

  return resource["resource_scopes"]

def getRearrangementUmaId(rearrangementId):
  repertoireId = cache.getRearrangementRepertoireId(rearrangementId):
  if repertoireId == None:
    repertoireId = repository.getRearrangement(rearrangementId)["repertoire_id"]
    cache.storeRearrangementRepertoireMapping(rearrangementId, repertoireId)

  return db.getRepertoireIdUmaId(repertoireId)
```

```
{
  "from": 1,
  "size": 2,
  "fields": [
    "repertoire_id",
    "subject.age"
  ],
  "include_fields": "miairr",
  "filters": {
    "op": "!=",
    "content": {
      "field": "repertoire_id",
      "value": "5e53dead4d808a03178c7891"
    }
  }
}
```

(a) User request

```
{
  "filters": {
    "op": "!=",
    "content": {
      "field": "repertoire_id",
      "value": "5e53dead4d808a03178c7891"
    }
  },
  "fields": [
    "study.study_id"
  ],
  "from": 1,
  "size": 2
}
```

(b) Modified Request

Figure 4.21: Example user request made to `POST /v1/repertoire` and it's modified version sent to the repertoire in order to emit the permission ticket. Notice how the `"fields"` parameter was modified.

```
# GET /v1/repertoire/:id
def getRepertoire(id, rptToken)
  umaId = db.getRepertoireIdUmaId(id)
  if umaId == None:
    raise 404

  tokenUmaResources = validateToken(rptToken, lambda: [umaId], getClassScopes("Repertoire"))
  streamFilteredResources(
    repository.getRepertoireStream(id),
    umaResourcesToGrantedFields(tokenUmaResources, "Repertoire"),
    "Repertoire",
    lambda rep: db.getStudyIdUmaId(rep["study"]["study_id"])
  )

# GET /v1/rearrangement/:id
def getRearrangement(id, rptToken)
  umaId = getRearrangementUmaId(id)
  if umaId == None:
    raise 404

  tokenUmaResources = validateToken(rptToken, lambda: [umaId], getClassScopes("Repertoire"))
  streamFilteredResources(
    repository.getRearrangementStream(id),
    umaResourcesToGrantedFields(tokenUmaResources, "Rearrangement"),
    "Rearrangement",
    lambda rearr: getRepertoireUmaId(rearr["repertoire_id"])
  )
```

For the ADC search endpoints (the POST endpoints excluding the `"facets"` function):

- `POST /v1/repertoire`: First the middleware validates the request body and checks that the request does not contain `"facets"` in the query.

  Then when no RPT token is supplied in the request the middleware modifies the request to perform the query in the repository and and only return the `"study.study_id"` field,



```
{
  "from": 1,
  "size": 20,
  "fields": [
    "sequence_id"
  ],
  "include_fields": "miairr",
  "filters": {
    "op": "!=",
    "content": {
      "field": "repertoire_id",
      "value": "12345"
    }
  }
}
```

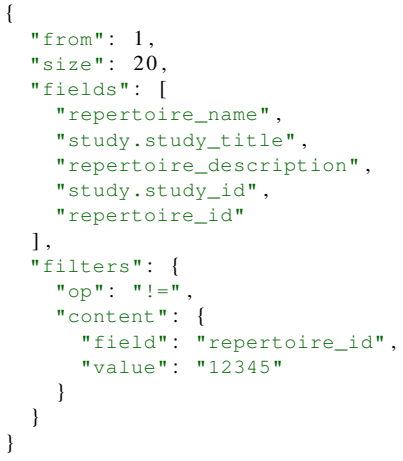
(a) User request



```
{
  "filters": {
    "op": "!=",
    "content": {
      "field": "repertoire_id",
      "value": "12345"
    }
  },
  "fields": [
    "sequence_id",
    "repertoire_id"
  ],
  "from": 1,
  "size": 20,
  "include_fields": "miairr"
}
```

(b) Modified Request

Figure 4.22: Example user request made to `POST /v1/rearrangement` and it's modified version sent to the repertoire when forwarding the request and returning the resources.



```
{
  "from": 1,
  "size": 20,
  "fields": [
    "repertoire_name",
    "study.study_title",
    "repertoire_description",
    "study.study_id",
    "repertoire_id"
  ],
  "filters": {
    "op": "!=",
    "content": {
      "field": "repertoire_id",
      "value": "12345"
    }
  }
}
```

Figure 4.23: Example user request for repertoire's public fields. The request is sent unmodified to the repository, without needing an RPT token. What are considered public fields depends on the CSV configuration file.

which is done by copying the `"from"`, `"size"` and `"filters"` parameters from the user's request and setting the mentioned field as the `"fields"` parameter, see figure 4.21 for an example. The result is then mapped to the set of UMA resource IDs that correspond to those resources. The scopes for the ticket are computed by inspecting the `"fields"`, `"include_fields"` and `"filters"` (using the `"field"` value of the filter objects) query parameters and mapping the requested fields to their scopes using the CSV configuration file. If the `"fields"` or `"include_fields"` parameters are not present all of the fields of the repertoire are considered. The permission ticket is then emitted using the mentioned IDs and set of scopes.

When an RPT token is present the user's request is analyzed to check whether the `"study.study_id"` repertoire field would be returned by the request, by inspecting the `"fields"` and `"include_fields"` query parameters, and if it would not be present the mentioned field is added to the `"fields"` parameter.

In figure 4.22 you can see an example of this for the search rearrangements endpoint. In the figure, the `"repertoire_id"` field was added because the user requested that this field be omitted. On the response to the user, this field will be filtered out.

The middleware then forwards the modified request, and streams back the result, filtering the returned resources. This filtering is achieved by introspecting the RPT token to obtain the list of resources and scopes to which the user can access, and then matching the returned resources, using the `"study.study_id"` field to the corresponding UMA ID and then checking using the scopes and the CSV configuration file which fields are to be filtered out (this does not filter out public fields) if all the fields are filtered out the empty resource is not returned. If the mentioned field was added to the request it is filtered out. The `"filters"` parameter is additionally checked, and if there are any fields in the query which are not present in a returned resource the resource is filtered out (including public fields). This is done to avoid information leaks since if a user uses, for example, the `"="` filter operator with a certain value and a resource is returned with only public information the user knows that the returned resource has the value he specified for the field in the filter operator (even if the field is protected).

For the full flow, 2 requests are made by the middleware to the repository for each complete user request.

The user can bypass the OIDC + UMA flow if he wants to only access public information anonymously. This is done by only specifying what are considered public fields for the repertoire in the `"fields"` and `"filters"` query parameter, see figure 4.23 for an example. This way, without using an RPT token, the user can request these public fields from the middleware, where the middleware would forward and stream back the request unmodified to the repository, using only one request to the repository for the full user request. The public fields for each resource type can be obtained by accessing the public fields endpoint

(endpoint B in table 4.4). This bypass of the flow will be called the public flow from now on, in contrast to the protected flow which is the flow that makes use of OIDC and UMA.

- `POST /v1/rearrangement`: The protection of this endpoint works the same as for the one above, but by using the rearrangement's `"repertoire_id"` field.

The search endpoint behaviour can be explained in more detail with the following pythonic pseudo-code:

```
from filtering import umaResourcesToGrantedFields, streamFilteredResources
from individual_endpoints import validateToken

repository = AdcClient("http://repository:80/airr/v1")
uma = UmaClient("http://Keycloak:8082")
db = DbRepository("http://postgres:5432")
cache = Cache("http://redis:6379")
fieldsCsv = csv.loadFile(config.load("app.adcCsvConfigPath"))

def getIncludeFields(className, includeFields):
  includes = None
  if includeFields == "miairr":
    includes = set("miairr")
  else if includeFields == "airr-core":
    includes = set("miairr", "airr-core")
  else if includeFields == "airr-schema":
    includes = set("miairr", "airr-core", "airr-schema")
  else
    return set()

  fields = fieldsCsv.select("field")
              .whereEquals("class", className)
              .whereIncluded("include_fields", includes)
  return set(fields)

def getFieldsScopes(className, fields):
  scopes = fieldsCsv.select("access_scope")
              .whereEquals("class", className)
              .whereEquals("protection", "protected")
              .whereIncluded("field", fields)
  return set(scopes)

def getClassFields(className):
  fields = fieldsCsv.select("field")
              .whereEquals("class", className)
  return set(fields)

def buildSearchRequestFields(className, requestBody):
  allRequestedFields = set(requestBody["fields"]) + getIncludeFields(className,
  ↪  requestBody["include_fields"])
  if allRequestedFields.empty():
    allRequestedFields = getClassFields(className)

def postValidateToken(requestBody, allRequestedFields, rptToken, className, resourceIdField,
↪  resourceIdsProducer):
  filtersFields = set(getAdcQueryFiltersFields(requestBody["filters"]))
  fields = allRequestedFields + filtersFields
```

```
  requestedScopes = getFieldsScopes(className, fields)
  if requestedScopes.empty(): # public fields access
    return []

  idsRequest = {
    "fields": [resourceIdField],
    "filters": requestBody["filters"],
    "from": requestBody["from"],
    "size": requestBody["size"],
  }

  return validateToken(rptToken, lambda: resourceIdsProducer(idsRequest), requestedScopes)

# requestBody and grantedFieldsMap are both modified by reference
def whitelistSearch(requestBody, grantedFieldsMap, allRequestedFields, resourceIdField,
↪  allRequestedFields):
  idFieldAlreadyRequested = allRequestedFields.contains(resourceIdField) or
  ↪  allRequestedFields.empty()
  if not idFieldAlreadyRequested:
    requestBody["fields"].add(resourceIdField)

  filtersFields = set(getAdcQueryFiltersFields(requestBody["filters"]))
  for _, fields in enumerate(grantedFieldsMap):
    if not (filtersFields - fields).empty():
      fields.removeAll()

    fields.remove(fields - allRequestedFields) # set intersection


def getRepertoireUmaIds(requestBody):
  repertoires = repository.searchRepertoires(requestBody)["Repertoire"]
  studyIds = [ rep["study"]["study_id"] for rep in repertoires ]
  return [ db.getStudyUmaId(studyId) for studyId in studyIds ]

def getRearrangementUmaIds(requestBody):
  rearrangements = repository.searchRearrangements(requestBody)["Rearrangement"]
  repertoireIds = [ rea["repetoire_id"] for rea in rearrangements ]
  return [ db.getRepertoireUmaId(repId) for repId in repertoireIds ]

# POST /v1/repertoire, not including facets
def searchRepertoires(requestBody, rptToken):
  validateAdcRequest(requestBody)
  if "facets" in requestBody:
    raise "error"

  allRequestedFields = buildSearchRequestFields("Repertoire", requestBody)
  tokenUmaResources = postValidateToken(requestBody, allRequestedFields, rptToken,
  ↪  "Repertoire", "study.study_id", getRepertoireUmaIds)
  grantedFieldsMap = umaResourcesToGrantedFields(tokenUmaResources, "Repertoire")
  whitelistSearch(requestBody, grantedFieldsMap, allRequestedFields, "study.study_id",
  ↪  allRequestedFields)

  streamFilteredResources(
    repository.searchRepertoiresStream(requestBody),
    grantedFieldsMap,
    "Repertoire",
    lambda rep: db.getStudyIdUmaId(rep["study"]["study_id"])
```

```
{
  "facets": "subject.age",
  "filters": {
    "op": "!=",
    "content": {
      "field": "repertoire_id",
      "value": "12345"
    }
  }
}
```

(a) User request

```
{
  "filters": {
    "op": "and",
    "content": [
      {
        "op": "in",
        "content": {
          "field": "study.study_id",
          "value": [
            "PRJNA248411"
          ]
        }
      },
      {
        "op": "!=",
        "content": {
          "field": "repertoire_id",
          "value": "12345"
        }
      }
    ]
  },
  "facets": "subject.age"
}
```

(b) Modified Request

Figure 4.24: Example user request made to `POST /v1/repertoire` with the facets function.

```
  )

# POST /v1/repertoire, not including facets
def searchRepertoires(requestBody, rptToken):
  validateAdcRequest(requestBody)
  if "facets" in requestBody:
    raise "error"

  allRequestedFields = buildSearchRequestFields("Rearrangement", requestBody)
  tokenUmaResources = postValidateToken(requestBody, allRequestedFields, rptToken,
  ↪  "Rearrangement", "repertoire_id", getRearrangementUmaIds)
  grantedFieldsMap = umaResourcesToGrantedFields(tokenUmaResources, "Rearrangement")
  whitelistSearch(requestBody, grantedFieldsMap, allRequestedFields, "repertoire_id",
  ↪  allRequestedFields)

  streamFilteredResources(
    repository.searchRearrangementsStream(requestBody),
    grantedFieldsMap,
    "Rearrangement",
    lambda rearr: getRepertoireUmaId(rearr["repertoire_id"])
  )
```

For the ADC facets endpoints (the POST endpoints including the `"facets"` parameter):

- `POST /v1/repertoire`: First the middleware validates the request body and checks that the request contains a `"facets"` parameter and does not contain a `"fields"` or `"include_fields"` query parameter.

  Then when no RPT token is present in the request the middleware performs the same steps as for the ADC search endpoints, that is: it modifies the request to obtain the list of repertoire's

```
{
  "facets": "subject.age"
}
```

(a) User request

```
{
  "filters": {
    "op": "in",
    "content": {
      "field": "study.study_id",
      "value": [
        "PRJNA248411"
      ]
    }
  },
  "facets": "subject.age"
}
```

(b) Modified Request

Figure 4.25: Example user request made to `POST /v1/repertoire` with the facets function. The added `"in"` filters operator makes sure the resources considered are the ones under the shown study (which is obtained from the UMA ID present in the RPT token).

```
{
  "from": 1,
  "size": 20,
  "facets": "repertoire_id",
  "filters": {
    "op": "!=",
    "content": {
      "field": "repertoire_id",
      "value": "12345"
    }
  }
}
```

Figure 4.26: Example user request made to `POST /v1/repertoire` for repertoire's facets using a public field (`"repertoire_id"`). The request is sent unmodified to the repository, without needing an RPT token. What are considered public fields depends on the CSV configuration file.

study IDs from the repository that match the user's filters, see figure 4.21 for an example. The scopes are obtained by reading the CSV configuration file to find the scopes for the fields in the `"facets"` and `"filters"` parameters.

When an RPT token is present the middleware introspects the token and obtains the list of UMA IDs. It then translates these UMA IDs to the set of repertoire IDs that belong to them and checks that the UMA ID's scopes match the joint scopes requested by the `"facets"` and `"filters"` parameters for each ID, and if they don't the ID is not used. The user's request body is then modified by inserting an `"in"` operator filter with the obtained list of repertoires as the value, see figure 4.25 (and an `"and"` root operator filter if the user had supplied his own filters, see figure 4.24), and the request is then forwarded to the repository and sent back to the user without any filtering. The `"in"` filter is what ensures that the resources being considered by the repository are resources the user has access to as specified in the RPT token. This approach with the `"in"` filter could also be used with the non-facets version of this endpoint to improve the performance on the second user request in the flow, though this is not implemented in the reference repository.

For the full flow 2 requests are made to the repository.

The user can bypass the OIDC + UMA flow as well by specifying public fields for the repertoire in the `"facets"` and `"filters"` query parameters, where the request would be forwarded unmodified by the middleware, see figure 4.26 for an example. The public fields endpoint can be used to obtain the list of public fields for each resource.

- `POST /v1/rearrangement`: The protection for this endpoint is the same as for the one above.

In figure 4.24 you can see how the user query's `"filters"` parameter is modified when requesting protected facets. The user had already supplied a filter so an `"and"` operator was added in addition to the `"in"` operator. The `"in"` operator makes sure the resources considered are the ones under the shown study (which is obtained from the UMA ID present in the RPT token).

The facets endpoint behaviour can be explained in more detail with the following pythonic pseudo-code:

```
from filtering import umaResourcesToGrantedFields, streamFilteredResources
from individual_endpoints import validateToken
from search import postValidateToken, getRepertoireUmaIds, getRearrangementUmaIds

repository = AdcClient("http://repository:80/airr/v1")
uma = UmaClient("http://Keycloak:8082")
db = DbRepository("http://postgres:5432")
cache = Cache("http://redis:6379")
fieldsCsv = csv.loadFile(config.load("app.adcCsvConfigPath"))

def facetsValidateToken(requestBody, rptToken, className, resourceIdField,
↪  resourceIdsProducer):
  idsRequest = {
```

```
    "fields": [resourceIdField],
    "filters": requestBody["filters"],
    "from": requestBody["from"],
    "size": requestBody["size"],
  }

  allRequestedFields = set(requestBody["fields"]) + getIncludeFields(className,
  ↪  requestBody["include_fields"])
  if allRequestedFields.empty():
    allRequestedFields = getClassFields(className)

  requestedScopes = getFieldsScopes(className, allRequestedFields)
  if requestedScopes.empty(): # public fields access
    return ([], allRequestedFields)

  tokenUmaResources = validateToken(rptToken, lambda: resourceIdsProducer(idsRequest),
  ↪  requestedScopes)
  return (tokenUmaResources, allRequestedFields)

# requestBody will be modified by reference
def addFacetsFilter(requestBody, resourceId, ids):
  inFilter = {
    "op": "in",
    "content": {
      "field": resourceId,
      "value": ids
    }
  }

  if "filters" in requestBody:
    andFilter = {
      "op": "and",
      "content": [
        inFilter,
        requestBody["filters"]
      ]
    }
    requestBody["filters"] = andFilter
  else:
    requestBody["filters"] = inFilter

# POST /v1/repertoire, including facets
def repertoiresFacets(requestBody, rptToken):
  validateAdcRequest(requestBody)
  if "facets" not in requestBody or "fields" in requestBody or "include_fields" in
  ↪  requestBody:
    raise "error"

  facetsField = requestBody["facets"]
  filtersFields = set(getAdcQueryFiltersFields(requestBody["filters"]))
  requestedFields = set(facetsField) + filtersFields
  tokenUmaResources = postValidateToken(requestBody, [ facetsField ], rptToken, "Repertoire",
  ↪  "study.study_id", getRepertoireUmaIds)
  if not tokenUmaResources.empty(): # non public field
    grantedFieldsMap = umaResourcesToGrantedFields(tokenUmaResources, "Repertoire")
    grantedUmaIds = [
      umaId for umaId, fields in enumerate(grantedFieldsMap) if
```

```
            umaId != None and requestedFields in fields
        ]

        nestedStudies = [db.getUmaResourceStudies(umaId) for umaId in grantedUmaIds]
        grantedStudies = [item for sublist in nestedStudies for item in sublist] # flatten

        addFacetsFilter(requestBody, "study.study_id", grantedStudies)

    return repository.repertoiresFacets(requestBody)

# POST /v1/rearrangement, including facets
def rearrangementsFacets(requestBody, rptToken):
    validateAdcRequest(requestBody)
    if "facets" not in requestBody or "fields" in requestBody or "include_fields" in
    ↪  requestBody:
        raise "error"

    facetsField = requestBody["facets"]
    filtersFields = set(getAdcQueryFiltersFields(requestBody["filters"]))
    requestedFields = set(facetsField) + filtersFields
    tokenUmaResources = postValidateToken(requestBody, [ facetsField ], rptToken,
    ↪  "Rearrangement", "repertoire_id", getRearrangementUmaIds)
    if not tokenUmaResources.empty(): # non public field
        grantedFieldsMap = umaResourcesToGrantedFields(tokenUmaResources, "Rearrangement")
        grantedUmaIds = [
            umaId for umaId, fields in enumerate(grantedFieldsMap) if
                umaId != None and requestedFields in fields
        ]

        nestedRepertoires = [db.getUmaResourceRepertoire(umaId) for umaId in grantedUmaIds]
        grantedRepertoires = [item for sublist in nestedRepertoires for item in sublist] #
        ↪  flatten

        addFacetsFilter(requestBody, "repertoire_id", grantedRepertoires)

    return repository.rearrangementsFacets(requestBody)
```

In this subsection, there have been references to the public fields endpoint (endpoint (B) in table 4.4). The CSV configuration file allows the repository admin to specify which fields are to be public, so a priori it is not known which fields are public for the repertoires or rearrangements, or if there are any. This endpoint returns the public fields for each resource type by reading the CSV configuration file. See figure 4.27 for an example response.

This endpoint can be described with the following pythonic pseudo-code:

```
from filtering import getClassPublicFields

fieldsCsv = csv.loadFile(config.load("app.adcCsvConfigPath"))

# GET /v1/public_fields
def public_fields():
    return {
        "Repertoire": getClassPublicFields("Repertoire"),
        "Rearrangement": getClassPublicFields("Rearrangement")
    }
```

```
{
    "Repertoire": [
        "repertoire_name",
        "study.study_title",
        "repertoire_description",
        "study.study_id",
        "repertoire_id"
    ],
    "Rearrangement": []
}
```

Figure 4.27: Examples response for the `GET /v1/public_fields` public endpoint.

Minimal ADC API Compliance

The middleware server supports arbitrary fields as previously mentioned if they are correctly configured in the CSV configuration file. Only a few fields are mandatory and hardcoded for each resource type and, excluding the facets functions of the search endpoints, the user requests are not modified by the middleware to enforce access control beyond avoiding the omission of certain fields. This means that there is a minimal ADC API that the middleware server supports:

- ADC Endpoints: Only the `POST /v1/repertoire` endpoint is really mandatory for access control, to be able to synchronize and filter responses, but the reference implementation has configuration restrictions making it mandatory to have all 4 endpoints implemented at least partially `GET v1/repertoire/{id}`, `GET v1/rearrangement/{id}`, `POST v1/repertoire` and `POST v1/rearrangement`.

- The POST endpoints: The repository can completely discard the user's request body for the requests `POST v1/repertoire` and `POST v1/rearrangement` and they must return a minimaly correct resource format. In this case the middleware facets function in the middleware must be disabled, by way of a configuration property, which will disable requests that have a `"facets"` query parameter. To enable facets support the repository must support the `"facets"` query parameter including the `"and"` and `"in"` filters operators correctly.

- ADC resources: Both repertoires and rearrangements can have arbitrary fields but must at least provide a few mandatory fields. For repertoires the fields `"repertoire_id"`, `"study.study_id"` and `"study.study_title"` must be present, though `"study.study_title"` is used for purely UI functionality. For rearrangements the fields `"repertoire_id"` and `"sequence_id"` must be present.

  See figure 4.28 for an example of minimally compliant responses.

. In the figure, (a) is a valid response for `GET /v1/repertoire/{id}` and `POST /v1/repertoire` and (b) is a valid response for `GET /v1/rearrangement/{id}` and `POST /v1/rearrangement`.

This is the case of the sciReptor repository's HTTP API, which was modified to expose a minimally compliant ADC API (without facets).

TSV format

```
{
  "Repertoire": [
    {
      "repertoire_id": "123adc",
      "study": {
        "study_id": "12",
        "study_title": "Research thingy"
      }
    }
  ]
}
```

(a) Minimal repertoire response schema

```
{
  "Rearrangement": [
    {
      "repertoire_id": "123adc",
      "sequence_id": "234"
    }
  ]
}
```

(b) Minimal rearrangement response schema

Figure 4.28: Examples of minimally compliant repertoire and rearrangement endpoint responses either for individual or search endpoints.

So far only the JSON format was discussed for the search endpoints, that is a `"json"` value for the `"format"` parameter (the default if omitted). In the reference implementation, the TSV format is not supported, and it is also not supported in any of the repositories under consideration (as of the time of writing). Support could be added by having the middleware request JSON from the repository and translating it to TSV itself, instead of parsing and filtering the TSV format as done for the JSON format.

Oracle Attack

An Oracle attack, a type of timing side-channel attack, while usually used to leak passwords can be used to expose private information by measuring the time it takes to receive HTTP responses to user requests (BB07, B07).

Because of the way access control is implemented for the POST endpoints an attacker can use an oracle attack to leak information from resource fields to which he doesn't have access. This can be done anonymously, without even needing to log in. An attacker can craft a specific query using the `"filters"` query parameter and then check the timings of the HTTP requests to check whether resources are matching his query.

The middleware, on requests to POST endpoints without an RPT token, when presented with a user query, will send the query to the repository unmodified (parameters `"from"`, `"size"` and `"filters"`) to obtain the list of resource IDs to emit a permission ticket. If the amount of resources returned by the repository is bigger the overall HTTP request from the user's perspective will take longer since more JSON data is returned to the middleware and IO is usually the most time-consuming operation. The attacker can send multiple HTTP requests with a different `"filters"` query parameter each time and the requests should take different amounts of time depending on if the amount of matching resources returned by the repository is different, which should scaly linearly with the amount.

There can be some jitter in the timings, from random network delays or other factors, so the attacker can send the same request multiple times and obtain the average which can be considered the "true" timing of the request.

The `"contains"` filters operator allows a user to partially match a string (for example using

a value `"a"` for field string `"cbatd"` would give a match). This allows an attacker to discover a field's value in less time since the number of HTTP requests needed to discover the value scales linearly with the size of the field's value (*num_possible_characters* times *length_field*) because when a match is made the attacker knows that a certain character is correct and when subsequent matches are made by iterating on the search value the adjacent characters can be found. To do this the attacker would try different values using this operator, for example for the fields `"repertoire_id"` which acts as an ID and `"sample.cell_phenotype"` which is the value that is to be discovered. The ID field would be discovered first, by trying different values until a match is found, that is, for values that do not match, no resources should be returned by the repository, for the value that matches the repository would return a resource to the middleware which should cause a noticeable difference in the time the HTTP request takes. Then the target value would be searched using the ID field with the `"and"` operator, to discover a field's value for the specific resource. Using this attack, all of the fields of all of the resources could be leaked.

A similar problem is found in the `"<"`, `">"`, `"<="` and `">="` operators which allows the attack to do a binary search to find numeric field values using a similar approach (scales logarithmicaly with the value).

The other operators are less dangerous since they only match a value exactly, that is the number of HTTP requests would scale exponentially with the length of the field's value (*num_possible_characters* to the power of *length_field*). This can still be dangerous if the field of a resource follows a pattern, such as an auto-incrementing ID field, or if the number of possible values for the field is small. The operators `"in"` and `"exclude"` are the most dangerous since they allow a user to match by equality multiple values in one request.

While this attack can be made unfeasible by disabling the `"filters"` parameter from POST requests, which the reference implementation allows doing using a configuration property, doing so would effectively disable the very useful searching function, therefore mitigations to this problem can be made, which while not completely eliminating this attack, would increase the cost of doing it, by increasing the number of HTTP requests needed to leak a field which might reach the point of becoming unfeasible. The mitigations that can be made are:

- Disabling the most problematic operators: As mentioned the `"contains"` makes the attack scale linearly and `"<"`, `">"`, `"<="`, `">="` make the attack scale logarithmically. Therefore these should be disabled by the middleware. The reference implementation disables these operators by default and also the `"in"` and `"exclude"` operators.

- Rate limiting: By rate limiting by the IP the number of HTTP requests that can be made per second the time needed to leak information increases. This can be mitigated by having a distributed set of computers attacking from different IPs, but this raises the cost of the attack. A reasonable limit can be arrived at which makes the API usable by regular users but frustrates attackers. The reference implementation does not provide rate limiting, and ideally, this should be enforced by a layer below the application layer.

- By making the responses from the user's perspective all take the same time. This would effectively defeat this attack, but implementing this mitigation is hard and is computationally wasteful and slow (which can be disastrous for usability depending on the slow down).

  One possible implementation of this is to make the repository return the full list of resources on each user request and then the middleware would apply the filtering parameters itself. This should effectively reduce the timings differences for the different number of matching resources but it would impose a far too considerable performance cost, especially when searching rearrangements.

- Changing how resource parent IDs are obtained: using the facets function instead of a regular resource search to obtain the resource IDs so that the middleware server can emit a permission ticket would be ideal since a facets search can return less information than a regular search, even returning the same amount of information with differing matching resources given that the parent ID would be used for the facets and there can be many resources under a parent resource. This can be an effective mitigation strategy, especially for rearrangements since in practice there are many rearrangements for each repertoire.

  This mitigation approach has the problem in that a facets search does not return the same information as a regular search when the `"from"` or `"size"` ADC parameters are used in the regular search since only the `"filters"` parameter can be used in conjunction with the `"facets"` parameter. This could still be used to obtain the parent IDs instead of a regular search when the mentioned parameters are not used in the request but that is easy to overcome for an attacker. To effectively use this approach the ADC API specification would have to change to allow the `"from"` and `"size"` parameters to be used with facets and behave the same as in a regular search or all of the protected repositories would have to implement this extension.

  This mitigation approach additionally has the problem that it increases the cost of becoming minimally ADC compliant, as sciReptor does not support the facets function, and adding this support would involve considerable work.

  This approach has the side benefit of speeding up this part of the flow since a facets response is usually faster than a regular search.

- Using the RPT token to obtain the list of parent IDs: when an RPT token is present on a regular search the middleware could modify the ADC query to add an `"in"` operator to only return the resources in the RPT, by mapping the UMA resource IDs to the corresponding resource's parent IDs (same mechanism as is used to protect facets).

  This is less important mitigation since an attacker would already have to be logged in to perform an attack in this stage of the flow.

- Adding a random jitter to the request: by making the thread handling the HTTP request sleep for a random amount of time it becomes harder to tell when a query is matching. This

can be solved by making more requests and averaging out this jitter but this increases the cost of an attack.

- Placing all of the components of the solution in the same physical server: having all of the components, especially the middleware and the repository in the same server would decrease the latency of the internal HTTP requests, which should create a smaller timing difference between the different matching requests which can increase attack costs by needing to average more requests to be certain of a timing difference.

Some of these mitigations that were not implemented in the reference implementation are practical to be implemented but were not because of time constraints. A combination of these mitigations could become effective in defeating this attack.

The `"size"` query parameter can be used similarly to leak the number of resources present in the repository, but this information is not considered important enough so no mitigations are considered for this case.

This attack is made more viable because a middleware server approach is taken to the problem of access control. Integrating security directly in the repository would make it harder to launch such an attack since the filtering of unwanted fields and resources could be performed in the database, which should create smaller timing differences which should be harder to tell apart.

DOS attack

A DOS (Denial of Service) attack is a type of attack that disables a server from servicing its clients (EBCS06, BCS06).

This attack can be, to some extent, made to any HTTP service as long as the cost is no issue. But given how the solution is implemented a DOS attack for the middleware is made more cost-effective than for the unprotected repository, especially with the `POST /v1/rearrangement` endpoint.

A user can send an empty request body to this endpoint, without an RPT token, thereby making the middleware request a lot of rearrangements to emit a small permission ticket (a rearrangements search can take in the order of seconds), which causes a great deal of internal network and CPU usage for the repository and middleware while causing little network and CPU usage for the attacker. An attacker could overwhelm the repository and middleware this way, although it is yet unknown how easily.

An effective mitigation for this attack is to rate limit by IP the number of requests permitted per second. An attacker would need to launch a DDOS attack to bypass this, which is more costly. Another mitigation is to scale up and horizontally the servers to be able to handle more requests per second, though this has monetary costs.

Discussion

This approach to the problem has some advantages:

- Requires no code change from ADC API compliant repositories.

- Improves code and concern decoupling.
- Can support all of the ADC API features, though in the reference implementation returning resources in TSV format is not supported.
- Non-compliant repositories can be protected by implementing a minimal ADC API.

And some limitations:

- The performance of this solution is untested. Especially for searching rearrangements, since this would involve making 2 expensive requests to the repository in the full flow.
- The middleware is another component that needs to be managed and maintained by the repository admin.
- The Oracle and DOS attacks as discussed above. Though they can be somewhat mitigated and it is yet unknown how viable and cost-effective it is to launch any of these attacks with the current reference implementation.

This approach will be validated in the next chapter.

4.5 Search Engine Integration

As mentioned it is expected that a researcher might in the future use a search engine to search multiple protected repositories.

Considering the architecture shown in figure 4.1 where ADC compliant repositories are protected by the middleware server and where for each of these repositories there is a distinct Keycloak instance running, a researcher to perform a cross repository search would craft his ADC search query, without specifying specific resource IDs in the `"filters"` ADC query parameter, and would send this request to the repository's search endpoint. For each request all of the steps of the flow as indicated in figure 4.4 need to be followed, that is, the researcher would log in using each of the repository's corresponding Keycloak instance, which can mean federated login using the same OpenID Connect identity such as the same ORCID account for each of them (if support for this is configured in that instance) and then the researcher would request the resources from each repository, either getting granted or denied access. This would have to be presented in the form of a web user interface given that the login step, using OpenID Connect, involves the use of HTTP redirects, making a web browser mandatory (at least for the login step). This solution with multiple Keycloak instances involves the laborious task, for the researcher, of performing login for each of the repositories, which can be done only once on session start since login tokens can be regenerated without researcher involvement. A single central Keycloak instance could be used that protects all of the repositories being searched allowing a user to log in only once (in the web UI) to perform the search, but this has drawbacks as discussed before, namely security issues.

For the repositories that don't expose an ADC API and where protection is integrated with the repository, such as in the case of immuneDB, the web UI that performs the search would need to

have an adapter to this repository, which might make it unfeasible to use the same search engine for different API types.

4.6 Permissions Dashboard

The permissions dashboard is responsible for allowing a resource owner to manage permissions to his resources and grant or deny access for them to researchers present in the system.

As mentioned Keycloak provides a ready-made permissions dashboard that can allow a resource owner to log in, and review what resources are being requested by which user and grant or deny them access, where the user can be an internal Keycloak account or an account associated with a third party OIDC IdP.

In Keycloak a resource owner is a regular user account (that can even access resources as a researcher) and a UMA resource in Keycloak can only be associated with one user at a time. Additionally once assigned to an owner the resource cannot be reassigned to a new user and it cannot be assigned using the admin UI but only when creating the resource through the UMA resource management API (though these are not UMA but Keycloak limitations). A resource owner can grant access only to a specific access level for a study (which corresponds to adding access to a specific set of UMA scopes for the UMA resource) and can remove or downgrade access (which corresponds to revoking scopes for a specific UMA resource), though in the provided Keycloak web UI only the UMA scopes are displayed and the resource owner needs to translate access levels to UMA scopes himself. A resource owner can also proactively share a resource with a researcher without him having ever attempted to access the resource.

A Keycloak admin can create and remove internal accounts for use by researchers. Additionally, he can manage the resources that are in the system (that correspond to studies) and change their name, additional attributes, which is useful for displaying in a UI.

As mentioned the creation and state synchronization of UMA resources in Keycloak is performed by the middleware server so no input from the admin is needed in this aspect, except to trigger the synchronization.

Given the limitations of the Keycloak permissions dashboard discussed here, some of the project's members are working on a new dashboard that tries to remove these limitations.

Chapter 5

Validation

In this chapter, the proposed solution, a middleware server protecting ADC v1 compliant repositories, is validated functionally (based on the requirements) and with the help of experts in the ADC API and the various repositories under consideration.

5.1 Requirements Validation

For the validation of functional requirements, a web frontend SPA was built for the middleware server (Rosc, osc), capable of making requests to all of the ADC API endpoints in the middleware and following both the protected and public flows for each endpoint. For the examples shown here the frontend was hosted on `localhost:3000`, middleware on `localhost:8080` and Keycloak on `localhost:8082`.

The full frontend has the single-page design as can be seen in 5.1. There are forms for making requests to all of the ADC API endpoints, with the user being able to modify some parameters for the POST requests and being able to login with Keycloak, using the login button in the top right corner. The red dot next to the login button indicates that the user is not logged in (anonymous) while a green dot indicates that the user is logged in and can make protected requests.

Additionally, the researcher's username used in these examples is the `user` and the resource owner's is the `owner`.

As part of the initial setup, the middleware's synchronization endpoint was triggered to synchronize the middleware's and Keycloak's state with the repository's state.

For the examples shown here, iReceptor Turnkey is used as the underlying repository.

5.1.1 Public Information

A researcher while not logged in (anonymous) can, as mentioned previously, access the ADC's `GET /v1` and `GET /v1/info` endpoints and the `POST /v1/repertoire` and `POST`

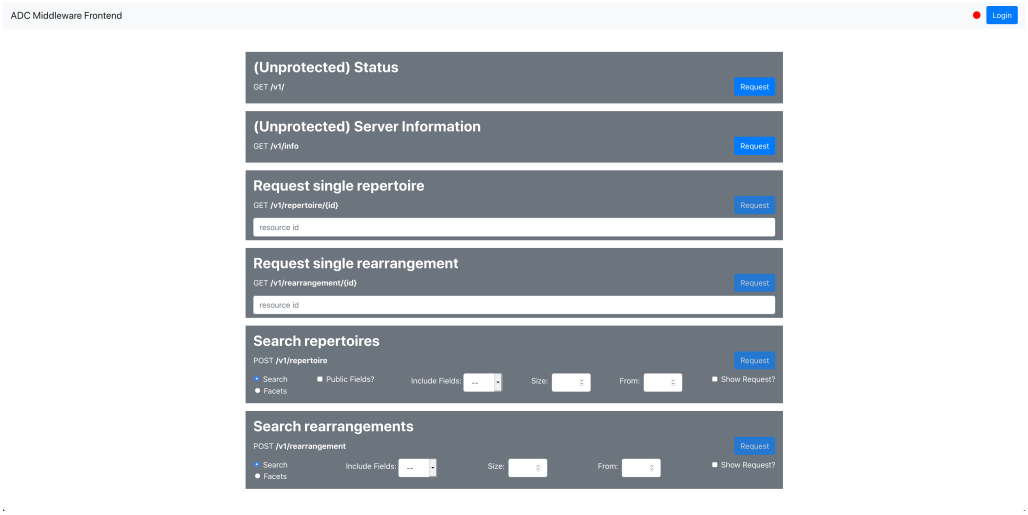


Figure 5.1: Front-end single page SPA overview.

`/v1/rearrangement` endpoints as long as he only uses public fields for the `"fields"`, `"include_fields"` `"filters"` and `"facets"` query parameters. The endpoints `GET /v1` and `GET /v1/info`, as can be seen in 5.2, are considered to not carry any information that needs to be protected so they are accessible by anyone.

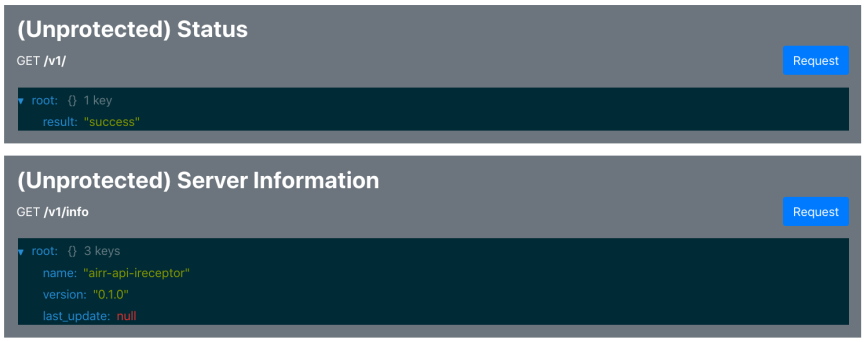


Figure 5.2: Public front-end endpoints `GET /v1` and `GET /v1/info` response examples.

See figure 5.3 for an example with regular search and figure 5.4 for an example with the facets function. In the figures, only public fields for the repertoire were requested and so the resources (and facets) were returned without emitting a permission ticket.

For the default configuration in use, no Rearrangement fields are configured as public, in the CSV configuration file, so the public flow for the `POST /v1/rearrangement` endpoint is disabled (any request would return a permission ticket or just return an error).

When a researcher adds a non-public field to his request for a POST endpoint the middleware automatically switches to the protected flow and returns a permission ticket along with an HTTP 401 response (no resources are returned). If any other errors occur the middleware returns an HTTP 401 error code without any additional information.

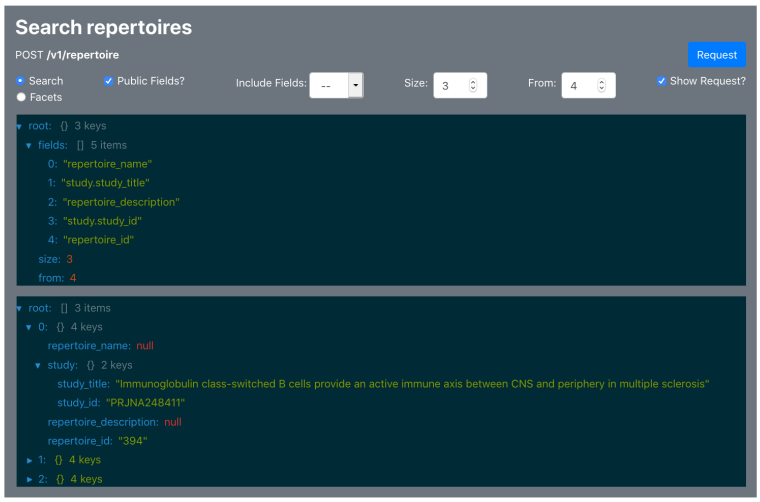


Figure 5.3: Front-end public regular search example request and response for `POST /v1/repertoire`.

If a researcher were to anonymously try to access the resources using the GET endpoints the middleware would automatically emit a permission ticket. Additionally, the same as for POST endpoints, on any error the HTTP 401 code is returned unless the resource doesn't exist in which case the HTTP 404 error code is returned.

5.1.2 Researcher Login

A researcher can log in by using a local Keycloak account or by using a third-party account, such as from EGI Check-In if support for the IdP was configured in the Keycloak instance.

As can be seen in figure 5.5, using the frontend the researcher can click the login button, in the top right corner, which will take the researcher's browser to Keycloak's user login page, where he can log in using a regular Keycloak account, as can be seen in figure 5.6.

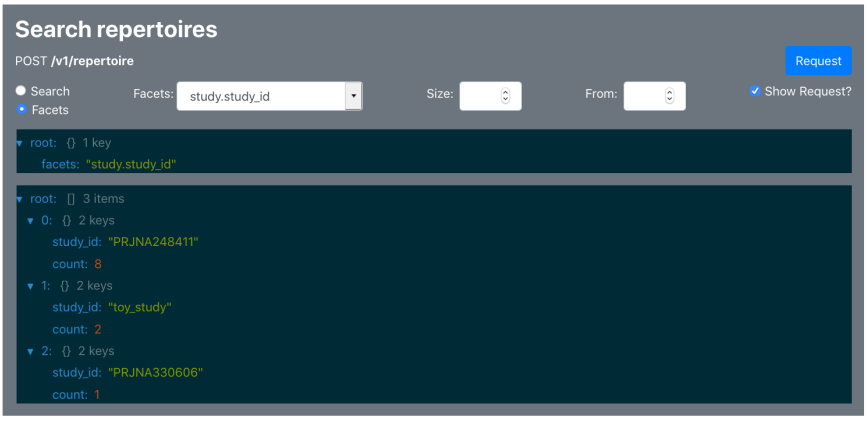


Figure 5.4: Front-end public facets search example request and response for `POST /v1/repertoire`.



Figure 5.5: Front-end login action example.

The researcher can also login using a third-party Identity Provider such as EGI Check-In, as can be seen in figure 5.7, which will redirect the user to the Identity Provider's login page, as seen in figure 5.8.

EGI Check-In acts as an IdP proxy allowing the user to login with any of the supported third-party IdPs as can be seen in the image. On successful login, in the IdP the user is redirected back to Keycloak. If the user is performing his first login with the IdP he is presented with a form page on Keycloak to provide his email and name, which will be associated with his IdP account and displayed on the permissions dashboard UI.

After a successful login in Keycloak, using either login flow, the researcher's browser is then redirected automatically back to the frontend where he is now logged in and can make protected requests, as can be seen in figure 5.9. If the user fails to login using invalid credentials, Keycloak will simply show an error on the login page without redirecting the user back to the frontend.

5.1.3 Resource Access Request

Once the researcher has logged in, he can request access to a resource owner's resources simply by requesting those resources.

In figure 5.10 you can see an example of a researcher without access to any resource requesting to access all of the repertoires in the repository. The user made a request to `POST /v1/repertoire` to access all of the resources and an alert showed up indicating that a request for these resources has been submitted for review by the resource owner.

The user can use any of the protected endpoints to request access. The middleware does not consume the researcher's login token to emit a permission ticket in this step, only when the researcher wants to obtain the RPT token is the login token used and the request for the resources submitted for review (done automatically by the frontend). If a user request for the POST endpoints would return an empty list of resources the middleware returns the HTTP 401 error code without any further information.

As can be seen in figure 5.11, the resource owner can review access requests for his resources by first logging in using Keycloak's login page for the realm (`/auth/realms/master/account` in the example's case for 'master' realm), and then accessing the permissions dashboard in the sidebar which allows him to then grant access, as can be seen in figure 5.12 which grants access to all of the scopes for the 2 studies.

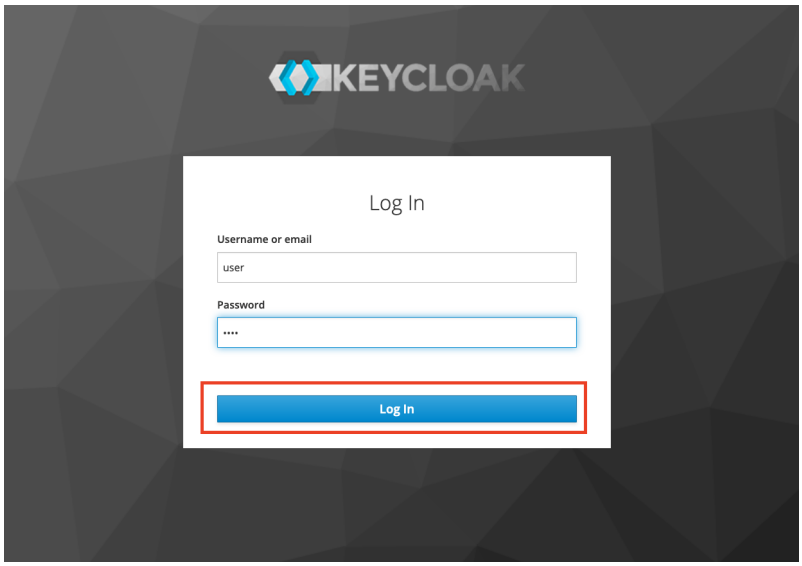


Figure 5.6: Keycloak regular user login action example.

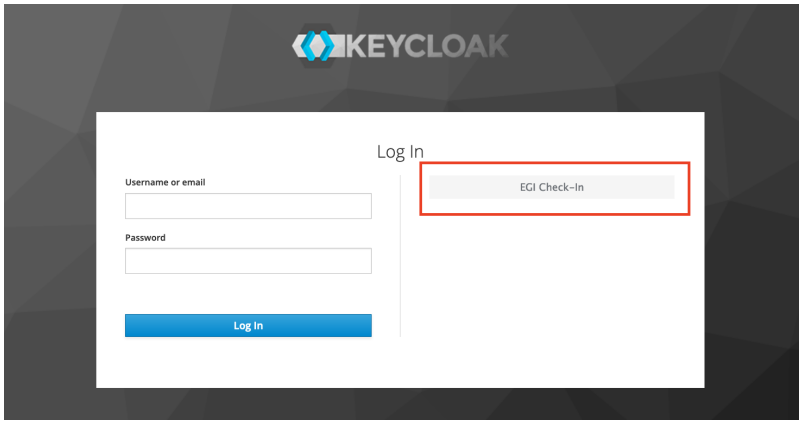


Figure 5.7: Keycloak third-party IdP login action example.



Figure 5.8: EGI Check-In login page example.

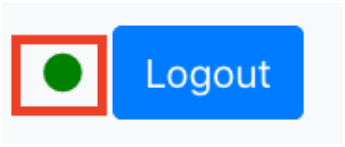


Figure 5.9: Front-end logged-in status UI section example.

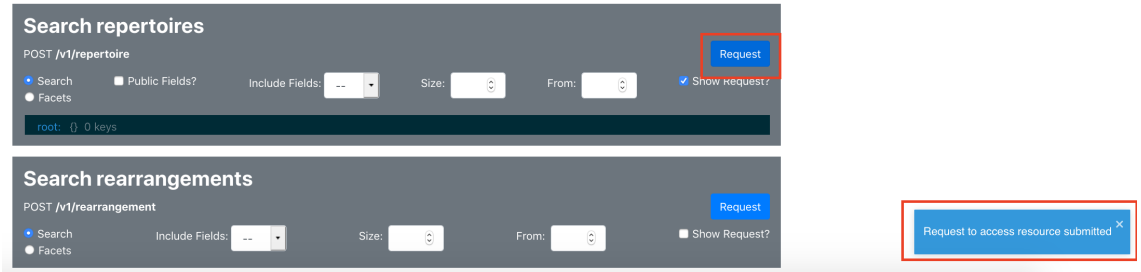


Figure 5.10: Front-end researcher resource request example.

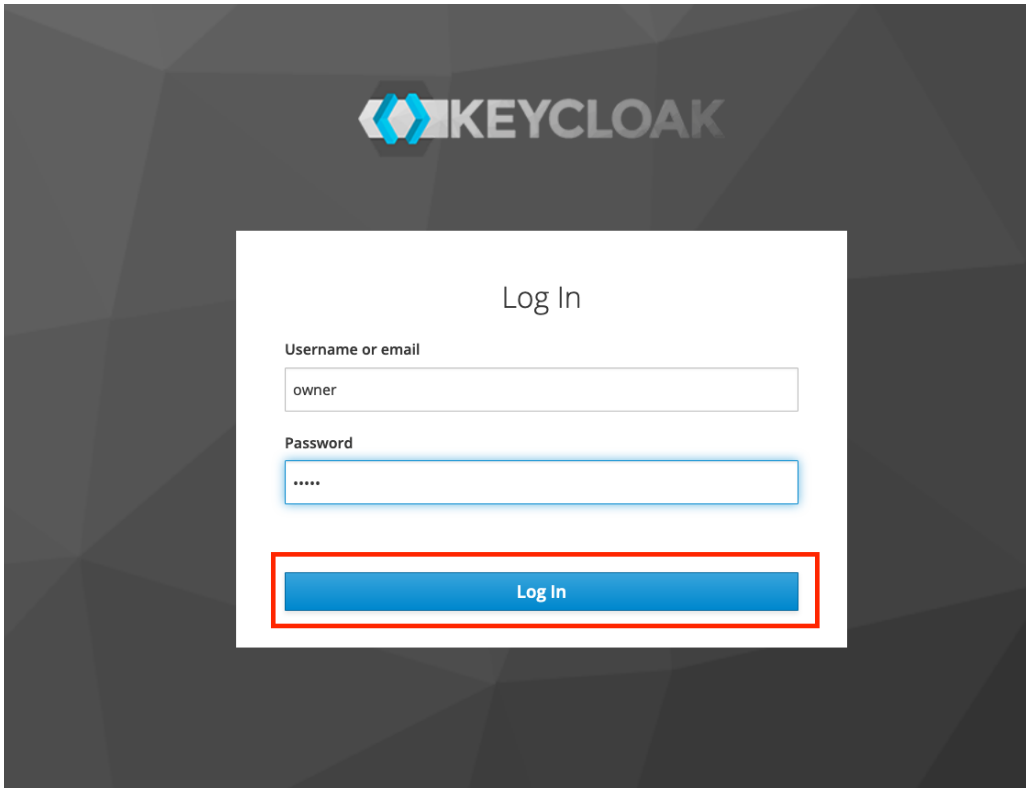


Figure 5.11: Keycloak resource owner's login action example.

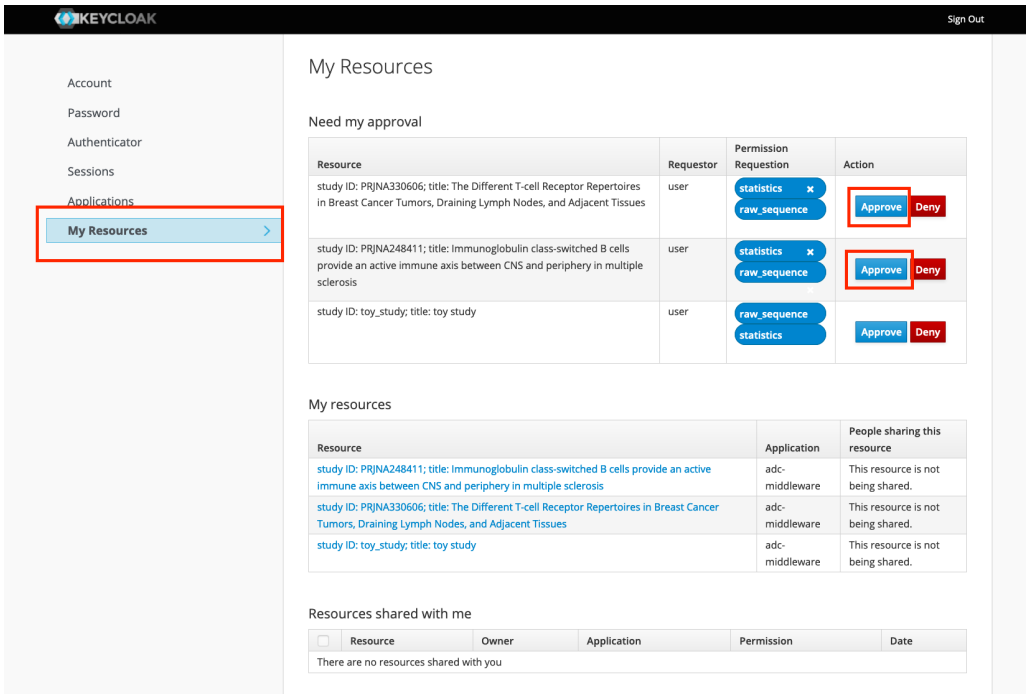


Figure 5.12: Keycloak permissions dashboard resource owner's access grant action example.

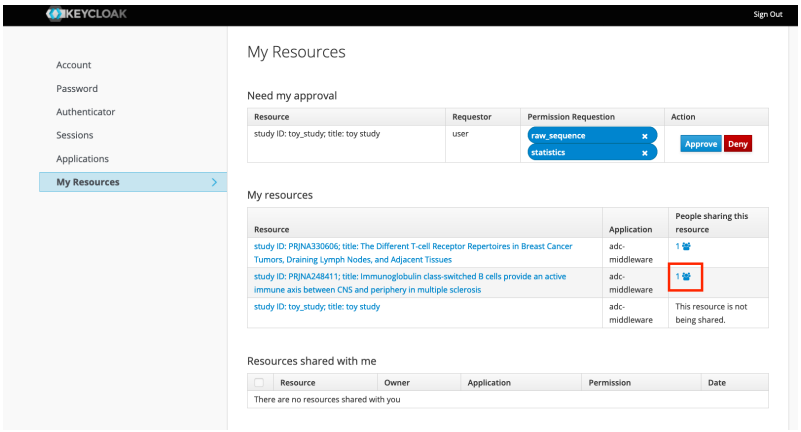


Figure 5.13: Keycloak permissions dashboard example action for accessing a single study.

A resource owner can also manage individual studies using the dashboard, as can be seen in figure 5.13, and revoke access to specific scopes or even the whole resource, as can be seen in figure 5.14.

In this page the resource owner can additionally preemptively grant access to a researcher for a study he owns, without the researcher having ever requested it, by specifying his account username or email, as can be seen in figure 5.15 where the owner grants access to all of the study's scopes for the `user` researcher.

As mentioned before, currently there can only be one resource owner in the system which will manage all of the resources for the repository. Additionally, the owner of a resource cannot change. The Keycloak account which will be the resource owner is set in the middleware's configuration file, and it is set when a resource is first created in Keycloak by way of the synchronize endpoint.

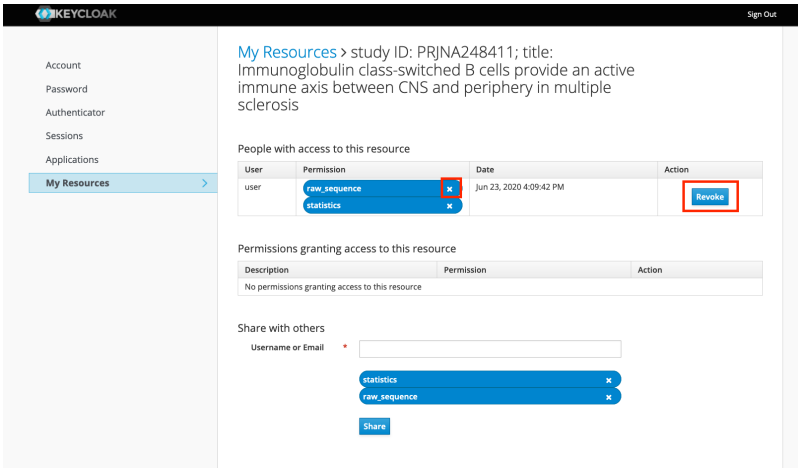


Figure 5.14: Keycloak permissions dashboard example action for revoking access.

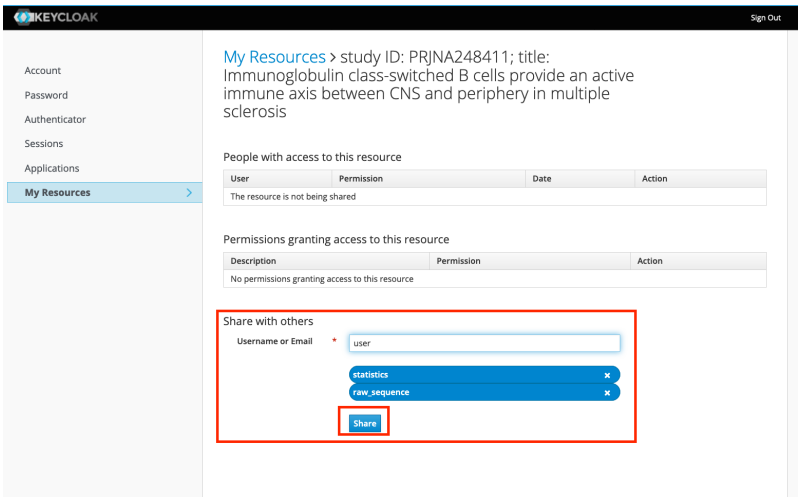


Figure 5.15: Keycloak permissions dashboard example action for sharing a resource.

5.1.4 Access Level Control

Here it is assumed that a researcher has been given full access to one study, partial access to another, and no access for the third and last study.

As can be seen in figure 5.16, the researcher can make a request to the `POST /v1/repertoire` endpoint (regular search) which will return filtered resources. Notice that the researcher requested all of the repertoires with all of the scopes (highest access level) but the middleware only returned the full fields for one repertoire (which corresponds to the study with full access, notice the presence of the `"sample"` field which is an access level above the other fields), filtered repertoires for others (which correspond to the study with partial access) and only public information for the remainder (which correspond to studies to which the user has no access). There are several repertoires with rearrangements in the repository used for this example.

This filtering also works for the individual GET endpoints, as can be seen in figure 5.17.

As can be seen in figure 5.18, access control based on granted scopes is also enforced for the POST facets function. Notice that the count returned is 1 for the field `"sample.cell_number"` which belongs to the highest access level, and that there are 11 repertoires in the repository from the previous examples which means that the counting was only performed on the one repertoire to which the researcher has access to with the specified access level in the facets field. The researcher's access to the other repertoires did not give him access to the requested field so these were not considered for counting by the repository.

Access control also works for the rearrangements, for individual GET (see figure 5.19), for regular POST search (see figure 5.20) and for the facets POST function (see figure 5.21). Notice that in the figures only the rearrangements for one repertoire were considered, given that all rearrangement fields have the highest access level, by default, and the researcher only has the highest access level for one repertoire.

For these requests, the RPT token is used to access the resources. If an RPT token is invalid

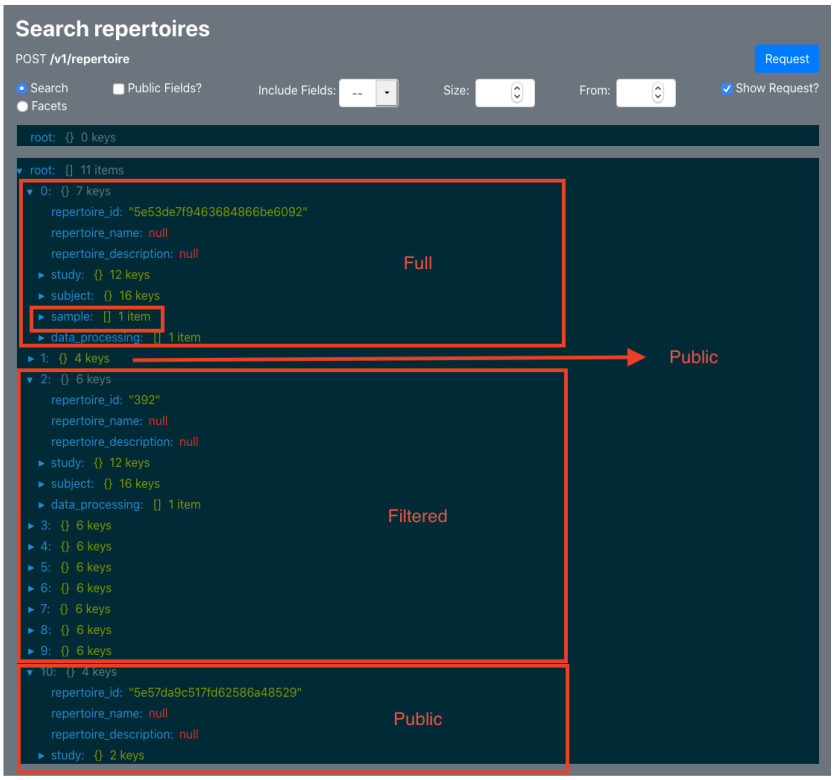


Figure 5.16: Front-end example repertoire regular search response with filtered repertoires.



Figure 5.17: Front-end example filtered single repertoire response. Notice the lack of the `"sample"` field which was filtered out.



Figure 5.18: Front-end example filtered repertoire facets response.

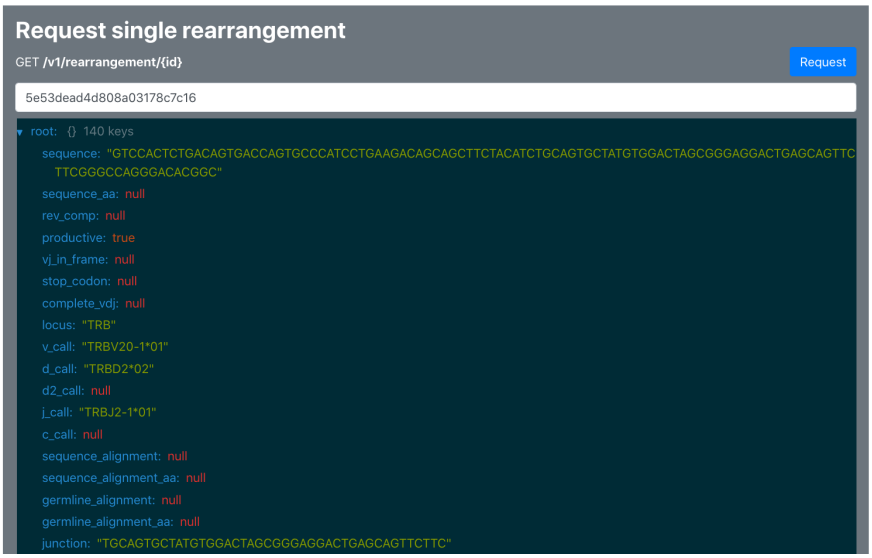


Figure 5.19: Front-end example single rearrangement response.

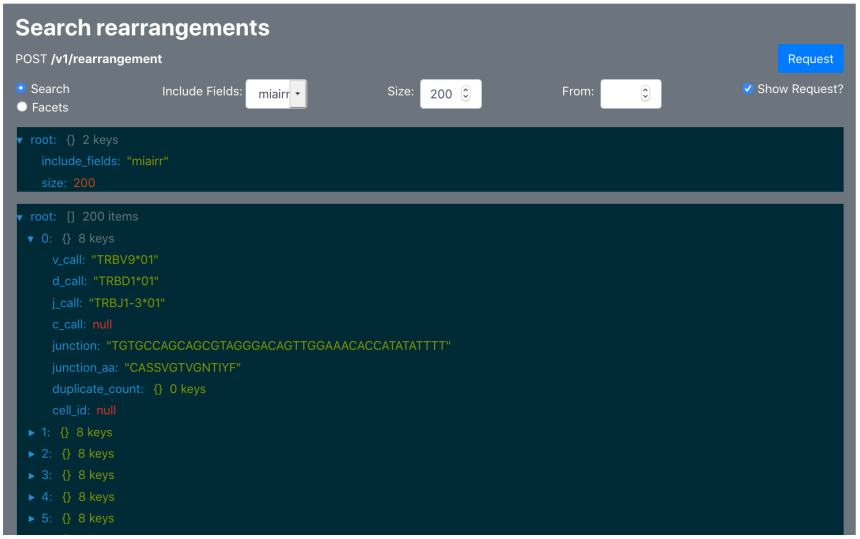


Figure 5.20: Front-end example filtered rearrangements regular search.

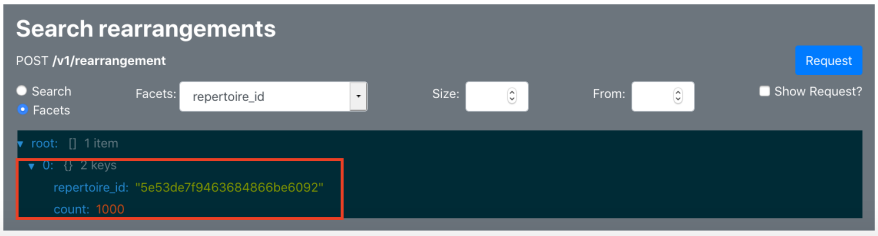


Figure 5.21: Front-end example filtered rearrangements facets search.

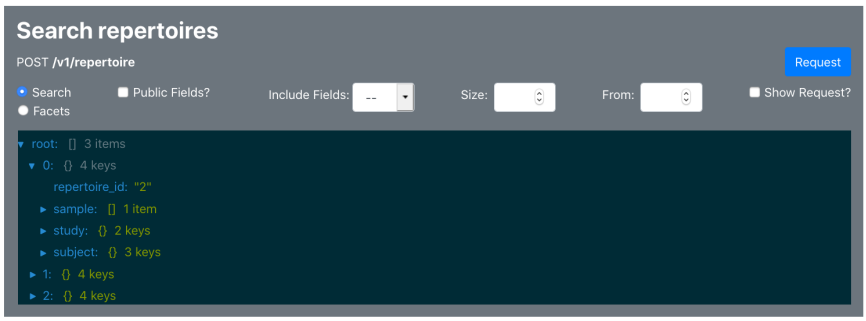


Figure 5.22: Front-end example sciReptor filtered repertoire regular search.

the middleware returns the HTTP 401 error code. The same error is returned when the middleware encounters any unexpected error.

If the RPT token does not have any resources with scopes that match the requested scopes in the facets request, no resources are considered for facets counting.

5.1.5 Multiple Repository Support

There is the requirement to support protection for both the iReceptor Turnkey and sciReptor repositories using this middleware solution. The protection of immuneDB falls outside the scope of the work for this dissertation.

As shown above, with the examples, fully ADC compliant repositories, like iReceptor Turnkey, can be protected with most of the repository's ADC features supported. In the case of Turnkey, all of the ADC features are supported since as of the time of writing this repository does not support TSV correctly.

In the case of sciReptor, the repository was updated to expose a minimally compliant ADC API, meaning that the GET and POST ADC endpoints are supported but the POST query parameters are discarded (including facets), in which case the facets function support in the middleware is disabled. Additionally, the ADC public endpoints are disabled in the middleware since they are not implemented in sciReptor. Fields filtering based on scopes and filtering based on requested fields (which is done in the middleware in this repository's case) is also supported.

You can see in figure 5.22 an example repertoire search, with a researcher's full access to all resources. Notice the reduced amount of repertoire fields compared to iReceptor Turnkey.

In figure 5.23 you can see an example with a researcher's partial access. Notice the lack of the `"sample"` field.

Fields filtering is supported as well for the rearrangement search and both individual endpoints.

Both the public and protected flow the POST endpoints are also supported.

5.1.6 Additional Requirements

The solution's performance regression on the endpoints (especially for the `POST /v1/rearrangement` endpoint) cannot be accurately validated until the repository main-

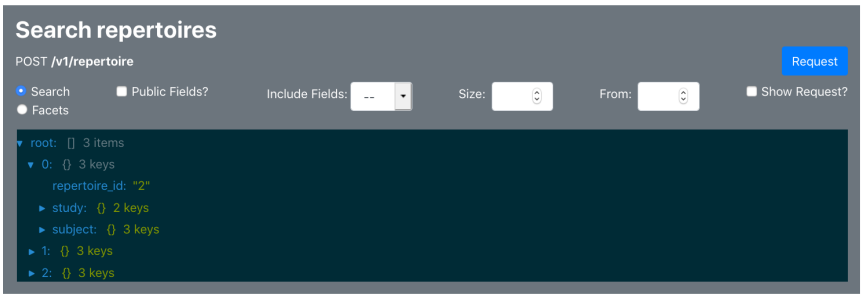


Figure 5.23: Front-end example sciReptor filtered repertoire facets search.

tainers integrate the solution with their production systems and test with some requests since the amount of data used in development for testing is nowhere near enough to know whether performance is still acceptable.

Additionally, while not a requirement, the synchronize endpoint was added to solve the problem of state synchronization between the middleware, Keycloack, and the repository. It is based on the assumption that the data in the repository is not updated frequently, which allows the synchronization to be performed in batch. This endpoint is password protected as mentioned before and it can be used as part of shell scripts.

5.2 Qualitative Analysis by Experts

A questionnaire was created and sent to 3 experts, which you can find in Appendix C. It was used to evaluate the viability of the solution from the perspective of repository maintainers and expert users of the ADC API:

- Brian Corrie: Responsible for the iReceptor Turnkey repository.
- Scott Christley: Responsible for the VDJ Server repository. VDJServer is an ADC compliant repository and it already has an integrated internal access control system.
- Christian Busse: Responsible for the sciReptor repository.

Additionally, all of the experts are members of the AIRR community knowledgeable in the AIRR's resource standards and the ADC API.

From their responses to the questionnaire, some insights were obtained.

5.2.1 Access Level Granularity

All of them agreed that a study-based access level granularity is acceptable, although Christian mentioned that a lover level might also be desirable such as per subject (repertoire).

Scott and Brian expressed the desire to be able to specify different access levels for the same fields but different studies, though to be able to support this use case each study would have to be configured manually which increases configuration complexity, and this is not currently compatible with the solution developed which would imply a lot of code change and re-design.

5.2.2 CSV configuration file

All of them agreed that it is useful to be able to edit and define additional custom fields in addition to those defined in the AIRR standard and that it is useful to be able to edit and define additional custom access levels for each field. They also mentioned that editing this file is not very user-friendly though it might be acceptable for an administrator to do so.

5.2.3 Synchronization

The mechanism used to trigger synchronization, a password protected endpoint, is perceived as not being user-friendly by Brian and Scott and as not being secure by Christian and Scott. They expressed concerns that the password might be sent in the clear or be stolen, which should be less of an issue if the middleware is running behind an SSL connection. As mentioned before this is not an optimal solution to protect this endpoint and some future work for this is discussed in the next chapter.

The synchronization process itself is perceived to be the right approach by Brian, since he states that data changes happen rarely in the iReceptor Turnkey repository, while Scott expresses that a true cache would be desirable, one that would not need to be previously synchronized and could create associations between IDs on the fly, though such a cache would increase complexity tremendously and would raise additional problems such as how and when to create UMA resources that match the studies.

Additionally, the limitation of there only being one resource owner (and that owner not being changeable after creation) for managing all the studies in the repository is addressed. They consider this to be a big limitation since in practice there can be multiple resource owners and there might be too many resources in the repository for only one person to manage, though they expressed that this limitation might be acceptable for smaller repositories. This is a limitation of Keycloak, and one solution would be to extend Keycloak to allow a resource's resource owner to be changed after it is created, such as by an admin, which could be done through the admin's web interface dashboard. Another solution would be to have the repository maintainer himself configure the UMA resources manually to set the desired resource owner, but this raises the problem of how to propagate the created UMA resource IDs to the middleware to create the association between UMA IDs and study IDs.

When synchronization occurs any resources in Keycloak that no longer have a match in the repository are deleted. This is considered a bad approach by Scott and Brian and they consider a better approach to mark the resources in Keycloak as inactive or deleted. This could be achieved by updating the UMA resource's name by adding a prefix instead of deleting the resource.

5.2.4 ADC endpoint protection

All agreed that resource field filtering is a good approach but Brian expressed the desire for the user to be able to tell which fields he was denied access to. This is simple to implement which can

be done by exposing an endpoint that returns all of the fields for each resource, though this might decrease security.

Christian expressed that some tools might break when some of the resource fields that are requested are not returned. This is a restriction of field-based filtering and might be fixed by returning an error code when access to a field is denied though this would imply a redesign of the access control solution since filtering is done in streaming mode.

All of them expressed that it is acceptable that a full flow to access resources might make 2 requests to the repository, though some reservations were made by Scott about the `POST /v1/rearrangement` endpoint since some of these queries might take a long time. Some performance improvements have been discussed in the previous chapter.

For the POST endpoints, a correction was made for information leaks which come from returning public fields for a resource that matches the user's query but that the user does not have access to for the queried fields, which simply involves filtering out a resource that matches that criteria (without even returning the public fields). All disagreed with this restriction since they expressed the desire for the user to know which resources match a query even if the user does not have access to those resources (by way of returning the resource's public fields). This is a limitation of this approach since returning these resources is a security issue. An approach to allow this use case needs to be studied, and it needs to be evaluated for feasibility and security since it would involve the re-design of the whole approach to access control taken here. Additionally for the side-channel attack version of this information leak (the Oracle Attack) the mitigation of disabling some dangerous operators was performed, which Scott disagreed with as he mentioned that this severely limits the functionality of the search endpoints, as the `"contains"` and `"in"` operators are widely used in practice. There are several other mitigations to this attack that were considered but not implemented because of time constraints and which are meant to replace this heavy-handed operator blacklisting approach.

5.2.5 Public information

All expressed that the endpoints `GET /v1`, `GET /v1/info`, `GET /v1/public_fields` can remain public and unprotected.

For the public flow of the POST endpoints, the approach to combine the `GET /v1/public_fields` endpoint and the ADC `"filters"` or `"facets"` parameters to access public information is considered acceptable by Brian and Christian.

5.2.6 ADC feature support

Both Brian and Scott mentioned that support for the TSV format is important since it is supposed to be the predominant format for rearrangements and is used by a lot of tools that target the AIRR standard. In the previous chapter, it was discussed how this feature might be implemented.

All agreed that the minimal ADC compliance is sufficiently flexible but Scoot expressed that it contains too many mandatory endpoints. As mentioned only the `POST /v1/repertoire`

endpoint is truly mandatory, additional configuration parameters could be added to the middleware to disable individually the other 3 ADC endpoints.

5.2.7 Overall Solution and architecture

Both Brian and Christian mentioned that a middleware server-based approach is desirable over integrating security directly in their repositories and that they intend to test and possibly integrate the solution in their respective repositories. Scott expressed that a middleware server solution is a good approach to security but that it is not flexible enough for VDJServer since it already has an integrated access control solution specific for their use cases.

All of them also agreed that a single Keycloak instance should be used for each repository and not a central one that manages multiple repositories.

Additionally, all of them perceive the overall solution to be secure and the usage of the ADC API to enforce the stated access control to be correct.

A concern raised by Brian is how to integrate third-party IdPs. Keycloak already supports this but additional documentation needs to be created on how to configure Keycloak to integrate with third-party IdPs like Egi Check-In.

Additionally Christian expressed the concern that some features from his repository might not be supported. This is a limitation of this approach to the problem since it only attempts to support ADC API features without implementing repository-specific extensions.

5.2.8 Additional Insights

From some additional discussions which were held with some potential users of the middleware server, the issue of integration with command line-like tools (without access to a browser) was raised. OAuth, and by extension OpenID Connect, mandates the use of a browser for user authentication as part of the protocol, although the middleware server is not the one responsible for this part of the flow but Keycloak. Keycloak has an extension endpoint that allows a user to obtain a login token by specifying the username and password in a regular HTTP request body, without needing a browser, so it should be possible to integrate this solution with purely command-line tools.

An additional issue raised in discussions is the use case of returning the studies which match an anonymous user's query using the public POST flow, even if the requested fields are protected, for the user to request access to these studies from the resource owner. The flow already supports this use case since middleware automatically requests access for these resources on behalf of the user by emitting a permission ticket when protected fields are requested which the user can try to exchange with Keycloak for an RPT token, which will place the requested resource for review by the respective resource owner. The only limitation of this approach to requesting access is that the user cannot know which studies match his query, only that some of them do.

Chapter 6

Conclusions

Healthcare data is more available than ever thanks to the spread of the Internet, so the job of a researcher who needs to access and process this data is easier than ever. But there are still many repositories with their own specific rules on how to access this data, many of them exposing publicly sensitive information.

An EU-Canadian project, called iReceptor+, was created where this thesis's work is inserted. And as part of this project an authentication and authorization system is proposed, allowing a researcher to use a single sign-on system with the EGI check-in proxy, whereby following the OpenID Connect protocol a researcher can obtain an ID token he can use to authenticate himself before systems protected with the authorization system. The goal is to have repositories that expose the ADC API standard to be protected by this system, allowing a researcher to access multiple repositories to which he was given access. This access control is achieved by having the researcher exchange the ID token with an access token, according to the UMA 2.0 protocol, which a researcher must present to the middleware reverse proxy, which handles access control on behalf of the ADC repositories, to access a resource of the API. The resource(s) returned by the repository are filtered by the middleware to enforce the access level restrictions as specified by the resource owners for their resources.

A system for managing the resource owners and their corresponding ownership of resources is also studied (the permissions dashboard) and so is the integration of protected repositories with this solution with a search engine.

The overall goal of protecting the iReceptor Turnkey and sciReptor repositories seems to have been achieved since their respective maintainers have shown interest in the middleware server solution and are already in the process of evaluating and testing it with the end goal of integrating the solution with their repositories in production.

6.1 Key Contributions

As part of the development of the reference implementation for this project several issues were identified in several open-source projects: Keycloak (Khod, hod) (Khob, hob) (Khoc, hoc) (Khoa, hoa), iReceptor Turnkey (Rosb, osb) (Rose, ose) and the AIRR's ADC API standard (Rosd, osd). Additionally, the sciReptor repository API was updated to expose a minimally compliant ADC API that could be protected by the reference implementation.

The reference implementation developed as part of the project supports enforcing information filtering based on user's access level, study-based access control granularity, synchronization of the Keycloak server with the repository, and supports protecting minimally ADC API compliant repositories. All of this without requiring any code change on behalf of the repositories.

Two of the overall project's repositories can be protected by this solution: iReceptor Turnkey and sciReptor (which was updated to expose a minimal ADC API). Others, such as immuneDB, can be protected if they migrate their HTTP API to ADC.

Integration with third-party IdPs such as EGI Check-in is shown to be supported, and also the integration with Keycloak's permissions dashboard for the management of resources by their resource owners.

Integration with a search engine capable of searching multiple repositories is also considered in the design of the solution.

6.2 Future Work

Because of time constraints, some issues remain open with the design and the reference implementation:

- Possibility of an oracle attack: an attack that can be carried out anonymously to leak information. Several mitigations are presented, with some already implemented in the reference implementation and some others that still need further development and implementation in some cases. Additionally, the monetary and time costs of making such an attack need to be studied to determine what are worthwhile mitigations to implement and what functionalities need to be sacrificed to make such an attack unfeasible.

  This is the most critical issue, though it does not prevent this solution from being usable given that the functionality exposing this attack, the use of the `"filters"` ADC parameter, can be completely disabled.

- Greater vulnerability to a DOS attack: given the design of the solution it becomes easier to conduct a DOS attack against the repository (and middleware) in contrast to a plain unprotected repository. A mitigation is discussed which should be considered for implementation.

- Support for ADC's TSV format: Support for filtering resources in TSV format is not yet supported by the middleware, being this the only ADC feature that is still not supported. The

implementation of this feature can be slow given that none of the repositories considered so far support this feature correctly.

- Limitation to only one resource owner: Because of Keycloak limitations and the way the synchronization process works only one resource owner manages all of the resources in Keycloak (studies) at any one time. An approach needs to be studied to allow multiple resource owners to manage their own set of resources, which might involve extending Keycloak.

- Support for future ADC endpoints: The ADC API specification is still in flux, and in the next major version, there are to be new endpoints dealing with statistics and clonotypes. When the new specification is officially released and there is support from some repositories the middleware should be updated to support these new endpoints.

- Improvement of the usage of the UMA flow: The middleware generates a new UMA PAT token on each request since the expiration time of the previous PAT tokens is not checked (which can be reused). Additionally, the introspection endpoint is called when the RPT tokens are JWS tokens which in themselves contain all of the required information, making this HTTP request unnecessary. Performing these optimizations should improve the performance of the requests.

- Performance validation: Given the design of the solution, many endpoints have worse performance than in the unprotected version, namely the `POST /v1/rearrangement` endpoint. Testing with production-level amounts of data needs to be performed to evaluate whether the performance regressions (compared to the unprotected repositories) are acceptable and if there is any value in improving the performance of the solution. Some possible performance improvements have already been discussed.

- Penetration testing: Because of time constraints a penetration test of the solution could not be performed by security experts, which would have been useful to identify potential security issues.

- Protection of the synchronize endpoint: currently, this endpoint is password protected which is stored as a hash in a configuration file. A better approach, by making use of a key management solution (such as that which might be provided by Keycloak), that can handle key revocation and replacement without a server restart needs to be evaluated.

Additionally, more repositories can be protected with this solution, such as immuneDB, by updating their API to be ADC compliant.

References

Alessandro Armando, Roberto Carbone, Luca Compagna, Jorge Cuellar, and Llanos Tobarra. Formal Analysis of SAML 2.0 Web Browser Single Sign-on: Breaking the SAML-Based Single Sign-on for Google Apps. In *Proceedings of the 6th ACM Workshop on Formal Methods in Security Engineering*, FMSE '08, page 1–10, New York, NY, USA, 2008. Association for Computing Machinery. doi:10.1145/1456396.1456397.

Rearrangement schema. Visited on 2020-06-05. URL: https://docs.airr-community.org/en/latest/datarep/rearrangements.html.

Repertoire schema. Visited on 2020-03-12. URL: http://docs.airr-community.org/en/latest/datarep/metadata.html.

Gail-Joon Ahn and Ravi Sandhu. Role-based authorization constraints specification. *ACM Trans. Inf. Syst. Secur.*, 3(4):207–226, November 2000. doi:10.1145/382912.382913.

Andrew Bortz and Dan Boneh. Exposing private information by timing web applications. In *Proceedings of the 16th International Conference on World Wide Web*, WWW '07, page 621–628, New York, NY, USA, 2007. Association for Computing Machinery. doi:10.1145/1242572.1242656.

BBC. Nhs data breach affects 150,000 patients in england. *BBC*, 2018. URL: https://www.bbc.com/news/technology-44682369.

Joseph Bonneau, Cormac Herley, Paul C. Van Oorschot, and Frank Stajano. Passwords and the evolution of imperfect authentication. *Communications of the ACM*, 58(7):78–87, 7 2015. doi:10.1145/2699390.

Ken Biba. Integrity considerations for secure computer systems. page 68, 04 1977. doi:10.13140/RG.2.2.11995.67369.

E. D. Bell and J. L. La Padula. Secure computer system: Unified exposition and multics interpretation, 1976. doi:10.21236/ada023588.

D. F. C. Brewer and Michael J. Nash. The chinese wall security policy. *Proceedings. 1989 IEEE Symposium on Security and Privacy*, pages 206–214, 1989. doi:10.1109/SECPRI.1989.36295.

Conor Cahill, John Hughes, Hal Lockhart, Michael Beach, Rebekah Metz, et al. Assertions and Protocols for the OASIS Security Assertion Markup Language (SAML) V2.0. page 86, 03 2005. URL: https://docs.oasis-open.org/security/saml/v2.0/saml-core-2.0-os.pdf.

Conor Cahill, John Hughes, Hal Lockhart, Michael Beach, Rebekah Metz, et al. Bindings for the OASIS Security Assertion Markup Language (SAML) V2.0. 03 2005. URL: http://docs.oasis-open.org/security/saml/v2.0/saml-bindings-2.0-os.pdf.

Conor Cahill, John Hughes, Hal Lockhart, Michael Beach, Rebekah Metz, et al. Metadata for the OASIS Security Assertion Markup Language (SAML) V2.0. 03 2005. URL: http://docs.oasis-open.org/security/saml/v2.0/saml-metadata-2.0-os.pdf.

Council of European Union. Regulation (eu) 2016/679 of the european parliament and of the council of 27 april 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing directive 95/46/ec (general data protection regulation), 2016. URL: https://op.europa.eu/en/publication-detail/-/publication/3e485e15-11bd-11e6-ba9a-01aa75ed71a1.

Khaled Elleithy, Drazen Blagovic, Wang Cheng, and Paul Sideleau. Denial of service attack techniques: Analysis, implementation and comparison. *Journal of Systemics, Cybernetics and Informatics*, 3:66–71, 01 2006.

EGI Check-in. Visited on 2020-01-31. URL: https://www.egi.eu/services/check-in/.

OpenID Connect Service Provider. Visited on 2020-01-31. URL: https://wiki.egi.eu/wiki/AAI_guide_for_SPs#OpenID_Connect_Service_Provider.

O. Ethelbert, F. F. Moghaddam, P. Wieder, and R. Yahyapour. A JSON token-based authentication and access management schema for cloud SaaS applications. In *Proceedings - 2017 IEEE 5th International Conference on Future Internet of Things and Cloud, FiCloud 2017*, volume 2017-January, pages 47–53, 2017. Cited By :3. doi:10.1109/FiCloud.2017.29.

David Ferraiolo and Richard Kuhn. Role-based access control. In *In 15th NIST-NCSC National Computer Security Conference*, volume arXiv:0903.2171, pages 554–563, 1992.

EGI Foundation. EGI Check-in service. Secure and user-friendly federated authentication and authorisation. Brochure, 2017. URL: https://www.egi.eu/wp-content/uploads/2017/09/Check-in.pdf.

Hardt. The OAuth 2.0 Authorization Framework. RFC 6749, Microsoft, October 2012.

N. Hossain, M. A. Hossain, M. Z. Hossain, M. H. I. Sohag, and S. Rahman. OAuth-SSO: A Framework to Secure the OAuth-Based SSO Service for Packaged Web Applications. In *Proceedings - 17th IEEE International Conference on Trust, Security and Privacy in Computing and Communications and 12th IEEE International Conference on Big Data Science and Engineering, Trustcom/BigDataSE 2018*, pages 1575–1578, 2018. doi:10.1109/TrustCom/BigDataSE.2018.00227.

Katharina Imkeller, Peter Arndt, Hedda Wardemann, and Christian Busse. sciReptor: Analysis of single-cell level immunoglobulin repertoires. *BMC Bioinformatics*, 17, 12 2016. doi:10.1186/s12859-016-0920-1.

IReceptor Plus. Visited on 2020-02-01. URL: https://www.ireceptor-plus.com/.

iReceptor Repositories. Visited on 2020-03-12. URL: http://ireceptor.irmacs.sfu.ca/repositories.

M. Jones and Microsoft. JSON Web Algorithms (JWA). RFC 7518, Internet Engineering Task Force (IETF), May 2015.

M. Jones, Microsoft, J. Bradley, Ping Identity, and N. Sakimura. JSON Web Signature (JWS). RFC 7519, Internet Engineering Task Force (IETF), May 2015.

M. Jones, Microsoft, J. Bradley, Ping Identity, and N. Sakimura. JSON Web Token (JWT). RFC 7519, Internet Engineering Task Force (IETF), May 2015.

M. Jones, Microsoft, and J. Hildebrand. JSON Web Encryption (JWE). RFC 7516, Internet Engineering Task Force (IETF), May 2015.

S. Josefsson. The Base16, Base32, and Base64 Data Encodings. RFC 4648, Network Working Group, October 2006.

Ed. J. Richer. OAuth 2.0 Token Introspection. RFC 7662, Internet Engineering Task Force (IETF), October 2015.

M. Jones, N. Sakimura, and J. Bradley. OAuth 2.0 Authorization Server Metadata. RFC 8414, Internet Engineering Task Force (IETF), June 2018.

Keycloack about. Visited on 2020-01-31. URL: https://www.keycloak.org/about.html.

Jan Kubovy, Christian Huber, Markus Jäger, and Josef Küng. A secure token-based communication for authentication and authorization servers. pages 237–250, 11 2016. doi:10.1007/978-3-319-48057-2_17.

Rostyslav Khoptiy. Can't use UMA PAT token for authorization on the introspection endpoint. Visited on 2020-06-15. URL: https://issues.redhat.com/projects/KEYCLOAK/issues/KEYCLOAK-13212.

Rostyslav Khoptiy. Endpoint for getting UMA resource is not UMA compliant. Visited on 2020-06-15. URL: https://issues.redhat.com/projects/KEYCLOAK/issues/KEYCLOAK-13209.

Rostyslav Khoptiy. 'token_type_hint' on token introspection endpoint is not UMA compliant. Visited on 2020-06-15. URL: https://issues.redhat.com/projects/KEYCLOAK/issues/KEYCLOAK-13211.

Rostyslav Khoptiy. When Updating an UMA resource the "name" is mandatory. Visited on 2020-06-15. URL: https://issues.redhat.com/projects/KEYCLOAK/issues/KEYCLOAK-14108.

Drexel University Systems Immunology Lab. immuneDB, 2018. Visited on 2019-11-19. URL: https://immunedb.readthedocs.io/en/latest/.

AIRR Data Commons API V1. Visited on 2020-02-03. URL: https://docs.airr-community.org/en/latest/api/adc_api.html.

Vladislav Mladenov, Christian Mainka, Julian Krautwald, Florian Feldmann, and Jörg Schwenk. On the security of modern Single Sign-On Protocols: OpenID Connect 1.0. *ArXiv*, abs/1508.04324, 2015.

Maciej Machulak and Justin Richer. Federated Authorization for User-Managed Access (UMA) 2.0. Draft Recommendation, Kantara Initiative, May 2017. URL:

https://docs.kantarainitiative.org/uma/wg/oauth-uma-federated-authz-2.0-05.html.

Maciej Machulak and Justin Richer. User-Managed Access (UMA) 2.0 Grant for OAuth 2.0 Authorization. Draft Recommendation, Kantara Initiative, November 2017. URL: https://docs.kantarainitiative.org/uma/wg/oauth-uma-grant-2.0-09.html.

Nitin Naik and Paul Jenkins. Securing digital identities in the cloud by selecting an apposite Federated Identity Management from SAML, OAuth and OpenID Connect. pages 163–174, 05 2017. doi:10.1109/RCIS.2017.7956534.

Department of Defense Standard. Department of defense trusted computer system evaluation criteria, 1985. doi:10.1007/978-1-349-12020-8_1.

Lawrence O'Gorman. Comparing passwords, tokens, and biometrics for user authentication. 12 2003. doi:10.1.1.200.3888.

orcid, openid connect, and implicit authentication.

ORCID about. Visited on 2020-01-31. URL: https://orcid.org/node/8.

Niels Provos and David Mazières. A future-adaptive password scheme. In *Proceedings of the Annual Conference on USENIX Annual Technical Conference*, ATEC '99, page 32, USA, 1999. USENIX Association. doi:10.5555/1268708.1268740.

Ross65536. ADC Middleware Server. commit b6b97c0f007b1341edd916285ec7a0cb8710e7c4. URL: https://github.com/Ross65536/adc-middleware.

Ross65536. Endpoint GET /rearrangement/<id> doesn't return correct ADC schema. Visited on 2020-06-15. URL: https://github.com/sfu-ireceptor/service-php-mongodb/issues/197.

Ross65536. Frontend for ADC middleware server. commit b33dbd670d6aeeae759757c5262e803af20adbb8. URL: https://github.com/Ross65536/adc-middleware-frontend.

Ross65536. Is the /v1/swagger endpoint deprecated? Visited on 2020-06-15. URL: https://github.com/airr-community/airr-standards/issues/415.

Ross65536. Is there going to be support for deprecation of rearrangement's 'rearrangement_id'? Visited on 2020-06-15. URL: https://github.com/sfu-ireceptor/service-php-mongodb/issues/207.

sciReptor. Visited on 2020-05-28. URL: https://scireptor-documentation.readthedocs.io/en/latest/.

R. Shirey. Internet Security Glossary, Version 2. RFC 4949, Network Working Group, August 2007.

P. Sriramya and R.A. Karthika. Providing password security by salted password hashing using bcrypt algorithm. *ARPN Journal of Engineering and Applied Sciences*, 10:5551–5556, 01 2015.

N. Sakimura, NRI, J. Bradley, Ping Identity, M. Jones, et al. OpenID Connect Core 1.0 incorporating errata set 1. Final Specification, OpenID Foundation, November 2014. URL: https://openid.net/specs/openid-connect-core-1_0.html.

VDJserver. Visited on 2019-11-19. URL: https://vdjserver.org/.

Appendix A

ADC API extension proposal

This document was created when the ADC API was called v1.3 (now currently called v1), the `rearrangement_id` and not `sequence_id` was the Rearrangement's main ID and the endpoint `GET /v1/swagger` existed.

A.1 Current API

This is the current ADC API as of the time of this writing:

Endpoints	Description
(1) GET /v1/repertoire/id	Retrieve a repertoire given its repertoire_id
(2) GET /v1/rearrangement/id	Retrieve a rearrangement given its rearrangement_id
(3) POST /v1/repertoire	Query/Search repertoires
(4) POST /v1/rearrangement	Query/Search rearrangements

Table A.1: ADC API overview, current standard (v1.3).

- (1) and (2) return individual resources as JSON objects.
- (3) and (4) are search endpoints allowing a user to perform a search. These return a list of JSON objects (the resources of (1) and (2)) that match the the user search query. The search query is sent in JSON form in the request body, containing keys like "from" and "size" that restrict the number of returned objects, "fields" that restrict what object/resource fields are to be returned, "filters" which contains the user search query, "facets" which returns aggregate counting information about a field, "format" which specifies whether to return data in JSON or AIRR TSV format (on POST) and "include_required" which specifies whether to return the required AIRR fields for the resource.

Endpoints $/v1, /v1/info$ and $/v1/swagger$ are not relevant for the proposed authorization design and can remain unchanged and public.

For examples see here.

A.2 Proposed ADC API extension

The main goal of the proposal is to make it simpler to implement an authentication and authorization scheme, based on OIDC and UMA, for resource servers that adopt this new API (the underlying backends) by providing a ready-made access control solution.

The new changes are to expose the same functionality as the previous API version but with the main idea of splitting the search (POST) endpoints into 2 endpoints each and of providing the full ID hierarchy for each individual resource.

It is intended that access control is managed by an intermediary middle-ware server (reverse proxy middleman) communicating with an UMA server, such as keycloak, and filtering the requests down to the resource server. The previous API would make it very impractical to have a middleware server filtering requests. Having a middle-ware server manage access control would allow any repository implementing this API extension get protection for free.

The following resources are identified in the new API: Studies which can contain multiple repertoires, repertoires which can contain multiple clonotypes and rearrangements. Additionally statistics endpoints are identified for repertoires and clonotypes.

Resources:

```
study -> repertoire ---> rearrangement/sequence
                     |-> clonotypes
```

As of this writing no APIs have been defined for clonotypes or statistics.

The new changes are based on the idea of studies as being the highest level resources, and therefore authorization is done around them. The levels of access are therefore defined for users for each individual study:

1. Level 1: Lowest level of access. Grants access to statistics about rearrangements and clonotypes. Corresponds to the UMA scope of statistics.

2. Level 2: Grants access to Level 1 resources and individual repertoires and clonotypes. Corresponds to UMA scopes of statistics and repertoire

3. Level 3: Highest level of access. Grants access to resources from levels 1 and 2 and individual rearrangements. Corresponds to UMA scopes of statistics, repertoire and sequences

A user has an access level for each individual study. If a user has no access level for a study he cannot access resources under it.

A user needs to have the same or higher access level than the endpoint's access level to be able to access the individual resource, for the specific study, unless using the indexing enpoints which will filter the returned fields according to the user's access level.

Some public endpoints are also defined, although they could still be protected under UMA (an UMA resource for each search endpoint) with the lowest access level needed.

If any resource as described exposed information from multiple levels of access the resource is considered to have the highest access level of the set.

The API is updated to be more RESTfull, though still not 100% RESTfull.

A.2.1 Why ADC v1.3 is impractical for a middle-ware server

A middle-ware server could theoretically be implemented for the current ADC v1.3 API to manage access control but it would be impractical, given that:

- Individual GET endpoints: Since access control is managed around studies each resource that is to be accessed still needs to have it's study ID presented to the middle-ware. For individual GET: (1) this would mean checking the returned study_id, for (2) this would mean making another request to get the repertoire mentioned in the repertoire to check the study ID. And given how UMA works this would mean wasted requests: first the user tries to access a repertoire resource without a token, the middleware server still needs to make the 2 requests to get the study ID to be able to emit a permissions ticket from the UMA authoriation server, then when the user makes a request to get the same resource with the RPT token the middleware server still needs to obtain the study ID again to check that the RPT grants access to that study ID, only then would the resource be returned.

  If a user makes an access to a resource to which he doesn't have access the middle-ware server would still need to make the requests to the backend to get the study ID.

  The performance could be improved by having a cache of ID mappings, but this would introduce the problem of synchronizing the middle-ware cache state with the back-end DB state. Any changes would have to be propagated upwards. If there are many changes to the backend keeping state synchronized could be a challenge.

  The proposed API extensions would need only one access to the backend in the UMA flow, given that the study ID would be present in the path.

- POST search endpoints: performance would be worse. To perform a search a user only specifies the filters query, and the resources and their IDs are only present in the response. For a search following the UMA flow, the first access without a token the middleware would have to send the request to the backend and parse the returned IDs in order to obtain the list of study IDs (without a cache, for a rearrangement this would imply another request to search the repositories) then a permissions ticket could be emitted (this request could be simplified with a "fields" to only return the resource ID), to obtain the resources themselves with the RPT another search request needs to emitted and results parsed and filtered (with a cache the filters query could be modified to make the resource IDs be in the IDs from the RPT token).

To allow field filtering based on access level on the final RPT request the middle-ware would have to decode and filter the json resources returned to only allow the pemitted fields for each access level related to each study, which could introduce ovearhead given that a search request could imply thousands of rearrangements (many MBs) which all need to be processed.

If the user introduced a specific “fields” value without the resource ID present, this would have to be added to the request and then filtered out by the middle-ware.

Using the API extensions no field filtering would be necessary, given that the middle-ware server could perform multiple backend requests, one for each access level, for public one set of resources with public “fields” , for level 1 another set of resources with level 1 “fields” , and so on.

The most performance friendly way to add UMA based access control to ADC v1.3 repositories is to add it directly for each repository, which implies implementing it for each repository (and accessing the databse and existing code, which can create bugs). A middleware “abstraction” is desired to manage access control to reduce the amount of work but is impractical for the current API, so extensions to the API are defined here which make implementing a middle-ware server practical in the sense of allowing acceptable performance and minimal overhead from the middle-ware.

A.2.2 New Endpoints

These endpoints contain breaking changes compared to v1.3 and therefore they need to be hosted under /v2 or some identifier other than /v1. The previous corresponding /v1 endpoints need to be denied external access for a repository as they are unprotected.

The access levels mentioned for each endpoint and the overall authentication and authorization system are irrelevant for the resource server given that access control is meant to be done through an intermediary middleware server.

While these endpoints use the full ID hierarchies for each resources, since access control is currently defined around studies only the study ID for each resource is sufficient for access control (e. g. can omit repertoire ID for rearrangements).

Endpoints	**Description**	**Access Level**
(a) GET /study/{id}/repertoire/{id}	Returns individual repertoire that belongs to the specified study	3
(b) GET /study/{id}/repertoire/{id}/rearrangement/{id}	Returns individual rearrangement that belongs to the specified study and repertoire	3
(c) GET /search/repertoire	Returns list of repertoire and study IDs matching user search	public
(d) GET /search/rearrangement	Returns list of rearrangement and study IDs matching user search	public
(O1) GET /repertoire	Returns repertoire resources by their URL as provided by user, and only to the ones the user has access.	2, 3, public
(O2) GET /rearrangement	Returns rearrangement resources by their URL as provided by user, and only to the ones the user has access.	2, 3, public

Table A.2: New ADC API proposal overview (v2).

- (a) returns the same objects as returned by (1) from /v1 except that the study ID is embedded in the path. It is very important that the resource server (backend) verifies that provided repertoire or rearrangement id belongs to the specified study ID in the path otherwise there can be a security issue.

  (a) is Level 3 and not 2 because repertoires can contain raw DNA sequences.

- Similarly (b) is the same as returned by (2). It is important that the resource server validates that the rearrangement belongs to the specified repertoire ID and the specified repertoire belongs to the specified study ID in order to avoid security issues.

- (c) and (d) correspond to (3) and (4) and are also search endpoints that behave mostly the same. The changes are: the “fields” , “include_required” , “format” and “facets” keys in the user JSON query are not allowed/discarded and the returned JSON is a list of resource URLs (for each hit the full IDs necessary to access the object individually is composed in the form of an URL). These endpoints can be made public.

- (O1) and (O2) accept a list of resource URLs in the request body in JSON form and returns for each URL the corresponding resource. The response resource order does not have to be the same as the request resource order. The response for each resource contains the URL identifier to allow matching of requests. The set of returned resources can be less than the set of requested resources, for example when a resource doesn’t exist, no public fields are allowed for resource, etc.

These endpoints are meant to complement the search endpoints. A user would use a search endpoint which is unprotected, and then pipe the result into this endpoint in order to enter the UMA flow.

The request URL are not sent as query params because of practical limitations on the URL path by some clients.

These endpoints are filtered by the server middleware, the middleware would unpack the resource URLs and perform the validation for each url and filter the requested URLs according to the user's access level for each study, potentially making multiple requests to the resource server, for each access level found.

The resource server would need to implement these endpoints, except for the managing of filtering according to the access level.

These endpoints can return some public information if specified in the "fields" without needing an UMA access token, for example for a list of repertoires return their corresponding study titles. Additionally these endpoints perform field filtering based on the access level for each study.

These endpoints would accept the "fields" , "include_required" , "format" and "facets" parameters.

- (O1) corresponds to (c). It might return some public fields when no RPT token is provided.
- (O2) corresponds to (d)
- (a) and (b) don't allow response field filtering based on access level, while (O1) and (O2) do allow.

See below for examples.

A.3 Examples

GET /study/{study_id}/repertoire/{repertoire_id}

Request

```
GET /study/PRJNA330606/repertoire/5e53de7f9463684866be6092
```

```
Accept: application/json
```

Response


```
{
    "Info": {
        "Title": "AIRR Data Commons authorization extension API",
        "description": "API response for repertoire query",
        "version": 1.0,
        "contact": {
            "name": "????",
            "url": "https://github.com/????"
        }
    },
    "Repertoire": [
        {
            "repertoire_id": "5e53de7f9463684866be6092",
            "study": {
                "study_id": "PRJNA330606",
                "study_title": "The Different T-cell Receptor Repertoires in Breast
                    ↪ Cancer Tumors, Draining Lymph Nodes, and Adjacent Tissues",
                "study_type": {
                    "value": "Case-Control Study",
                    "id": "C15197"
                },
                "inclusion_exclusion_criteria": "",
                "grants": "",
                "collected_by": " liuxiao@genomics.cn, vascular@fmmu.edu.cn",
                "lab_name": "Department of Vascular and Endocrine Surgery",
                "lab_address": "Beijing Genomics Institute (BGI) Xijing Hospital, Fourth
                    ↪  Military Medical University Medical University, Xi'an, China",
                "submitted_by": "T. Wang, C.Wang, J. Wu, C. He, X.Liu",
                "pub_ids": "http://cancerimmunolres.aacrjournals.org/content/5/2/148.
                    ↪ long",
                "keywords_study": [
                    "contains_tcr"
                ],
                "study_description": "Cancer Study"
            },
            ...
        }
    ]
}
```

GET/study/{study_id}/repertoire/{repertoire_id}/rearrangement/{rearrangement_id}

Request

```
GET /study/PRJNA330606/repertoire/5e53de7f9463684866be6092
    /rearrangement/5e53dead4d808a03178c7831

Accept: application/json
```

Response


```
{
    "Info": {
        "Title": "AIRR Data Commons authorization extension API",
        "description": "API response for repertoire query",
        "version": 1.0,
        "contact": {
            "name": "????",
            "url": "https://github.com/????"
        }
    },
    "Rearrangement": {
        "locus": "TRB",
        "v_call": "TRBV3-1*01",
        "d_call": "",
        "j_call": "TRBJ1-6*01",
        "c_call": null,
        "junction": "TGTGCCAGCAGCCAAGATTTAAATCTAGGGAACGGCCTATTTTCACCCCTCCACTTT",
        "junction_aa": "CASSQDLNLGNGLFSPLHF",
        "duplicate_count": "",
        "cell_id": null,
        "clone_id": null,
        "rearrangement_id": "5e53dead4d808a03178c7891",
        "repertoire_id": "5e53de7f9463684866be6092",
        "sample_processing_id": "5e53de7f9463684866be6092",
        "data_processing_id": "5e53de7f9463684866be6092",
        "v_score": 451,
        "d_score": null,
        "j_score": 139,
        "sequence": "
            ↪ CTAAATCTCCAGACAAAGCTCACTTAAATCTTCACATCAATTCCCTGGAGCTTGGTGACTCTGCTGTGTATTTCTGTGCCAGCAGCCAAGATTTA
            ↪ ",
        "sequence_id": "SRR4084215.182 FCC3J1LACXX:1:1101:20844:1797 length=76",
        "junction_aa_length": 19,
        "junction_length": 57,
        "productive": true
    }
}
```

GET /search/repertoire

Request

Notice the lack of “fields” in the JSON request.

```
GET /search/repertoire

Accept: application/json
Content-Type: application/json

{
    "filters":{
        "op":"and",
        "content": [
            {
                "op":"=",
                "content": {
                    "field":"subject.organism.id",
                    "value":"9606"
                }
     },
     {
                "op":"=",
                "content": {
                    "field":"sample.pcr_target.pcr_target_locus",
                    "value":"TRB"
                }
     }]
    },
    "from":10,
    "size":5
}
```

Response

```
[
  "/v2/study/PRJNA330606/repertoire/5e53de7f9463684866be6092",
  "/v2/study/PRJNA330607/repertoire/5e53de7f9463684866be6093",
  ...
]
```

GET /search/rearrangement

Request

Notice the lack of “fields” in JSON request.

```
GET /search/rearrangement

Accept: application/json
Content-Type: application/json
```


```
{
    "filters":{
        "op":"and",
        "content": [
            {
                "op":"in",
                "content": {
                    "field":"repertoire_id",
                    "value":["5e53de7f9463684866be6092","5e53de7f9463684866be6093"]
                }
        },
        {
                "op":"=",
                "content": {
                    "field":"junction_aa",
                    "value":"CARDPRSYHAFDIW"
                }
        }]
    },
    "from":10,
    "size":5
}
```


Response


```
[
"/v2/study/PRJNA330606/repertoire/5e53de7f9463684866be6092/rearrangement/5e53dea
    ↪ d4d808a03178c7833",
"/v2/study/PRJNA330607/repertoire/5e53de7f9463684866be6093/rearrangement/5
    ↪ e53dead4d808a03178c7834",
   ...
]
```

GET /repertoire

Request

The request can also contain “fields” , “include_required” , “facets” and “format” json values, and they work as in the previous API version. See examples below

```
GET /repertoire

Accept: application/json
Content-Type: application/json
```


```
{
    "urls": [
        "/v2/study/PRJNA330607/repertoire/5e53de7f9463684866be6092",
        "/v2/study/PRJNA330606/repertoire/5e53de7f9463684866be6093",
        "/v2/study/PRJNA330606/repertoire/5e53de7f9463684866be6094"
    ]
}
```


Response

Notice that the number returned is less than the number requested, the out of order response and the filtered public fields in the last response, form the user not having access to the study.

Notice also the new “Urls” returned field which maps 1-to-1 to the resources returned in “Repertoire” and maps these resources to the requested path.


```
{
    "Info": {
        "Title": "AIRR Data Commons authorization extension API",
        "description": "API response for repertoire query",
        "version": 1.0,
        "contact": {
            "name": "????",
            "url": "https://github.com/????"
        }
    },
    "Repertoire": [
        {
            "repertoire_id": "5e53de7f9463684866be6094",
            "study": {
                "study_id": "PRJNA330606",
                "study_title": "The Different T-cell Receptor Repertoires in Breast
                    ↪ Cancer Tumors, Draining Lymph Nodes, and Adjacent Tissues",
                "study_type": {
                    "value": "Case-Control Study",
                    "id": "C15197"
                },
                "inclusion_exclusion_criteria": "",
                "grants": "",
```

```
                "collected_by": " liuxiao@genomics.cn, vascular@fmmu.edu.cn",
                "lab_name": "Department of Vascular and Endocrine Surgery",
                "lab_address": "Beijing Genomics Institute (BGI) Xijing Hospital, Fourth
                    ↪  Military Medical University Medical University, Xi'an, China",
                "submitted_by": "T. Wang, C.Wang, J. Wu, C. He, X.Liu",
                "pub_ids": "http://cancerimmunolres.aacrjournals.org/content/5/2/148.
                    ↪ long",
                "keywords_study": [
                    "contains_tcr"
                ],
                "study_description": "Cancer Study"
            },
            ...
        },
        {
            "repertoire_id": "5e53de7f9463684866be6092",
            "study": {
                "study_id": "PRJNA330607",
                "study_title": "Immunoglobulin class-switched B cells provide an active
                    ↪ immune axis between CNS and periphery in multiple sclerosis"
            }
        }
    ],
    "Urls": [
        "/v2/study/PRJNA330606/repertoire/5e53de7f9463684866be6094",
        "/v2/study/PRJNA330607/repertoire/5e53de7f9463684866be6092"
    ]
}
```


More examples of requests:

- Facets

Request:


```
{
    "facets": "subject.sex",
    "urls": [
        "/airr/v2/study/PRJNA330606/repertoire/5e53de7f9463684866be6092",
        "/airr/v2/study/toy_study/repertoire/5e68d4c206e94e498f47a732",
        "/airr/v2/study/PRJNA248411/repertoire/392",
        "/airr/v2/study/PRJNA248411/repertoire/393",
        "/airr/v2/study/PRJNA248411/repertoire/394",
        "/airr/v2/study/PRJNA248411/repertoire/395",
        "/airr/v2/study/PRJNA248411/repertoire/396",
        "/airr/v2/study/PRJNA248411/repertoire/397",
        "/airr/v2/study/PRJNA248411/repertoire/398",
        "/airr/v2/study/PRJNA248411/repertoire/399",
        "/airr/v2/study/toy_study/repertoire/5e57da9c517fd62586a48529"
    ]
}
```


Response:

The total sum of count might be different from the number of requested resources

```
{
    "Info": {
        "Title": "AIRR Data Commons authorization extension API",
        "description": "API response for repertoire query",
        "version": 1.0,
        "contact": {
            "name": "????",
            "url": "https://github.com/????"
        }
    },
    "Facet": [
        {
            "subject.sex": "M",
            "count": 2
        },
        {
            "subject.sex": "Male",
            "count": 4
        },
        {
            "subject.sex": "Female",
            "count": 5
        }
    ]
}
```


- Fields

Request:


```
{
    "fields": [
        "repertoire_id",
        "subject.sex"
    ],
    "urls": [
        "/airr/v2/study/PRJNA330606/repertoire/5e53de7f9463684866be6092",
        "/airr/v2/study/toy_study/repertoire/5e68d4c206e94e498f47a732"
    ]
}
```


Response:


```
{
    "Info": {
        "Title": "AIRR Data Commons authorization extension API",
        "description": "API response for repertoire query",
        "version": 1.0,
        "contact": {
            "name": "????",
            "url": "https://github.com/????"
        }
    },
    "Repertoire": [
        {
            "repertoire_id": "5e53de7f9463684866be6092",
```

```
            "subject": {
                "sex": "Female"
            }
        },
        {
            "repertoire_id": "5e68d4c206e94e498f47a732",
            "subject": {
                "sex": "M"
            }
        }
    ],
    "Urls": [
        "/airr/v2/study/PRJNA330606/repertoire/5e53de7f9463684866be6092",
        "/airr/v2/study/toy_study/repertoire/5e68d4c206e94e498f47a732"
    ]
}
```


- Include Required

Works similar to “fields” . “include_required” : false is the same as omitting.

Request:


```
{
    "include_required": true,
    "urls": [
        "/airr/v2/study/PRJNA330606/repertoire/5e53de7f9463684866be6092",
        "/airr/v2/study/toy_study/repertoire/5e68d4c206e94e498f47a732"
    ]
}
```


Response:

Same as for “fields”.

- Format

A value for “format” of “json” is the same as omitting this field.

Example for a value of “tsv” :

Request:


```
{
    "format": "tsv",
    "urls": [
        "/airr/v2/study/PRJNA330606/repertoire/5e53de7f9463684866be6092",
        "/airr/v2/study/toy_study/repertoire/5e68d4c206e94e498f47a732"
    ]
}
```


Response:

A “url” column would be added to the returned tsv for each resource, containing the full URL for that resource.

GET /rearrangement

Request

The request can also contain “fields” , “facets” , “include_required” and “format” json values, and they work as in the previous API version. See usage examples in the previous request.

```
GET /rearrangement

Accept: application/json
Content-Type: application/json
```


```
{
    "urls": [
        "/v2/study/PRJNA330606/repertoire/5e53de7f9463684866be6092/rearrangement/5e53dea
            ↪ d4d808a03178c7833",
        "/v2/study/PRJNA330606/repertoire/5e53de7f9463684866be6092/rearrangement/5e53dea
            ↪ d4d808a03178c7834"
    ]
}
```


Response

Notice that the number returned is less than the number requested.


```
{
    "Info": {
        "Title": "AIRR Data Commons authorization extension API",
        "description": "API response for repertoire query",
        "version": 1.0,
        "contact": {
            "name": "????",
            "url": "https://github.com/????"
        }
    },
    "Rearrangement ": [
        {
            "locus": "TRB",
            "v_call": "TRBV9*01",
            "d_call": "TRBD1*01",
            "j_call": "TRBJ1-3*01",
            "c_call": null,
            "junction": "TGTGCCAGCAGCGTAGGGACAGTTGGAAACACCATATATTTT",
            "junction_aa": "CASSVGTVGNTIYF",
            "duplicate_count": "",
            "cell_id": null,
            "clone_id": null,
            "rearrangement_id": "5e53dead4d808a03178c7831",
            "repertoire_id": "5e53de7f9463684866be6092",
            "sample_processing_id": "5e53de7f9463684866be6092",
            "data_processing_id": "5e53de7f9463684866be6092",
```

```
            "v_score": 260,
            "d_score": 35,
            "j_score": 135,
            "sequence": "
                ↪ GAGCTCTCTGGAGCTGGGGGACTCAGCTTTGTATTTCTGTGCCAGCAGCGTAGGGACAGTTGGAAACACCATATATTTTGGAGAG
                ↪ ",
            "sequence_id": "SRR4084215.4 FCC3J1LACXX:1:1101:1558:1921 length=74",
            "junction_aa_length": 14,
            "junction_length": 42,
            "productive": true
        }
    ],
    "Urls": [
         "/v2/study/PRJNA330606/repertoire/5e53de7f9463684866be6092/rearrangement/5
             ↪ e53dead4d808a03178c7833"
    ]
}
```


The examples for “facets” , “fields” , “include_required” , “format” are similar to the previous endpoint examples.

Appendix B

Fields Access Level Configuration File

This file is the current, as of the time of writing, fields access level CSV configuration file:

```
class,field,protection,access_scope,field_type,include_fields,description
Repertoire,repertoire_id,public,,string,airr-core,"Identifier for the repertoire object.
    ↪ This identifier should be globally unique so that repertoires from multiple
    ↪ studies can be combined together without conflict. The repertoire_id is used to
    ↪ link other AIRR data to a Repertoire. Specifically, the Rearrangements Schema
    ↪ includes repertoire_id for referencing the specific Repertoire for that
    ↪ Rearrangement."
Repertoire,repertoire_name,public,,string,airr-schema,Short generic display name for the
    ↪ repertoire
Repertoire,repertoire_description,public,,string,airr-schema,Generic repertoire
    ↪ description
Repertoire,study.study_id,public,,string,miairr,Unique ID assigned by study registry
Repertoire,study.study_title,public,,string,miairr,Descriptive study title
Repertoire,study.study_type.value,protected,statistics,string,miairr,Type of study
    ↪ design
Repertoire,study.study_type.id,protected,statistics,string,miairr,Type of study design (
    ↪ Ontology ID)
Repertoire,study.study_description,protected,statistics,string,airr-schema,Generic study
    ↪ description
Repertoire,study.inclusion_exclusion_criteria,protected,statistics,string,miairr,List of
    ↪ criteria for inclusion/exclusion for the study
Repertoire,study.grants,protected,statistics,string,miairr,Funding agencies and grant
    ↪ numbers
Repertoire,study.collected_by,protected,statistics,string,miairr,"Full contact
    ↪ information of the data collector, i.e. the person who is legally responsible for
    ↪ data collection and release. This should include an e-mail address."
Repertoire,study.lab_name,protected,statistics,string,airr-core,Department of data
    ↪ collector
Repertoire,study.lab_address,protected,statistics,string,airr-core,Institution and
    ↪ institutional address of data collector
Repertoire,study.submitted_by,protected,statistics,string,miairr,"Full contact
    ↪ information of the data depositor, i.e. the person submitting the data to a
    ↪ repository. This is supposed to be a short-lived and technical role until the
    ↪ submission is relased."
Repertoire,study.pub_ids,protected,statistics,string,miairr,Publications describing the
    ↪ rationale and/or outcome of the study
```

```
Repertoire,study.keywords_study,protected,statistics,array_string,miairr,Keywords
    ↪ describing properties of one or more data sets in a study
Repertoire,subject.subject_id,protected,statistics,string,miairr,"Subject ID assigned by
    ↪  submitter, unique within study"
Repertoire,subject.synthetic,protected,statistics,boolean,miairr,TRUE for libraries in
    ↪ which the diversity has been synthetically generated (e.g. phage display)
Repertoire,subject.organism.value,protected,statistics,string,airr-schema,Binomial
    ↪ designation of subject's species
Repertoire,subject.organism.id,protected,statistics,string,airr-schema,Binomial
    ↪ designation of subject's species (Ontology ID)
Repertoire,subject.sex,protected,statistics,string,miairr,Biological sex of subject
Repertoire,subject.age_min,protected,statistics,number,miairr,Specific age or lower
    ↪ boundary of age range.
Repertoire,subject.age_max,protected,statistics,number,miairr,Upper boundary of age
    ↪ range or equal to age_min for specific age. This field should only be null if
    ↪ age_min is null.
Repertoire,subject.age_unit.value,protected,statistics,string,miairr,Unit of age range
Repertoire,subject.age_unit.id,protected,statistics,string,miairr,Unit of age range (
    ↪ Ontology ID)
Repertoire,subject.age_event,protected,statistics,string,miairr,"Event in the study
    ↪ schedule to which `Age` refers. For NCBI BioSample this MUST be `sampling`. For
    ↪ other implementations submitters need to be aware that there is currently no
    ↪ mechanism to encode to potential delta between `Age event` and `Sample collection
    ↪  time`, hence the chosen events should be in temporal proximity."
Repertoire,subject.age,protected,statistics,string,airr-schema,DEPRECATED
Repertoire,subject.ancestry_population,protected,statistics,string,miairr,Broad
    ↪ geographic origin of ancestry (continent)
Repertoire,subject.ethnicity,protected,statistics,string,miairr,Ethnic group of subject
    ↪ (defined as cultural/language-based membership)
Repertoire,subject.race,protected,statistics,string,miairr,Racial group of subject (as
    ↪ defined by NIH)
Repertoire,subject.strain_name,protected,statistics,string,miairr,Non-human designation
    ↪ of the strain or breed of animal used
Repertoire,subject.linked_subjects,protected,statistics,string,miairr,Subject ID to
    ↪ which `Relation type` refers
Repertoire,subject.link_type,protected,statistics,string,miairr,"Relation between
    ↪ subject and `linked_subjects`, can be genetic or environmental (e.g.exposure)"
Repertoire,subject.diagnosis.study_group_description,protected,statistics,string,miairr,
    ↪ Designation of study arm to which the subject is assigned to
Repertoire,subject.diagnosis.disease_diagnosis,protected,statistics,string,miairr,
    ↪ Diagnosis of subject
Repertoire,subject.diagnosis.disease_length,protected,statistics,string,miairr,Time
    ↪ duration between initial diagnosis and current intervention
Repertoire,subject.diagnosis.disease_stage,protected,statistics,string,miairr,Stage of
    ↪ disease at current intervention
Repertoire,subject.diagnosis.prior_therapies,protected,statistics,string,miairr,List of
    ↪ all relevant previous therapies applied to subject for treatment of `Diagnosis`
Repertoire,subject.diagnosis.immunogen,protected,statistics,string,miairr,"Antigen,
    ↪ vaccine or drug applied to subject at this intervention"
Repertoire,subject.diagnosis.intervention,protected,statistics,string,miairr,Description
    ↪  of intervention
Repertoire,subject.diagnosis.medical_history,protected,statistics,string,miairr,Medical
    ↪ history of subject that is relevant to assess the course of disease and/or
    ↪ treatment
Repertoire,sample.sample_id,protected,raw_sequence,string,miairr,"Sample ID assigned by
    ↪ submitter, unique within study"
```

```
Repertoire,sample.sample_type,protected,raw_sequence,string,miairr,"The way the sample
    ↪ was obtained, e.g. fine-needle aspirate, organ harvest, peripheral venous
    ↪ puncture"
Repertoire,sample.tissue,protected,raw_sequence,string,miairr,"The actual tissue sampled
    ↪ , e.g. lymph node, liver, peripheral blood"
Repertoire,sample.anatomic_site,protected,raw_sequence,string,miairr,"The anatomic
    ↪ location of the tissue, e.g. Inguinal, femur"
Repertoire,sample.disease_state_sample,protected,raw_sequence,string,miairr,
    ↪ Histopathologic evaluation of the sample
Repertoire,sample.collection_time_point_relative,protected,raw_sequence,string,miairr,"
    ↪ Time point at which sample was taken, relative to `Collection time event`"
Repertoire,sample.collection_time_point_reference,protected,raw_sequence,string,miairr,
    ↪ Event in the study schedule to which `Sample collection time` relates to
Repertoire,sample.biomaterial_provider,protected,raw_sequence,string,miairr,Name and
    ↪ address of the entity providing the sample
Repertoire,sample.tissue_processing,protected,raw_sequence,string,miairr,Enzymatic
    ↪ digestion and/or physical methods used to isolate cells from sample
Repertoire,sample.cell_subset.value,protected,raw_sequence,string,miairr,Commonly-used
    ↪ designation of isolated cell population
Repertoire,sample.cell_subset.id,protected,raw_sequence,string,miairr,Commonly-used
    ↪ designation of isolated cell population (Ontology ID)
Repertoire,sample.cell_phenotype,protected,raw_sequence,string,miairr,List of cellular
    ↪ markers and their expression levels used to isolate the cell population
Repertoire,sample.cell_species.value,protected,raw_sequence,string,airr-schema,"Binomial
    ↪  designation of the species from which the analyzed cells originate. Typically,
    ↪ this value should be identical to `organism`, if which case it SHOULD NOT be set
    ↪ explicitly. Howver, there are valid experimental setups in which the two might
    ↪ differ, e.g. chimeric animal models. If set, this key will overwrite the `
    ↪ organism` information for all lower layers of the schema."
Repertoire,sample.cell_species.id,protected,raw_sequence,string,airr-schema,"Binomial
    ↪ designation of the species from which the analyzed cells originate. Typically,
    ↪ this value should be identical to `organism`, if which case it SHOULD NOT be set
    ↪ explicitly. Howver, there are valid experimental setups in which the two might
    ↪ differ, e.g. chimeric animal models. If set, this key will overwrite the `
    ↪ organism` information for all lower layers of the schema. (Ontology ID)"
Repertoire,sample.single_cell,protected,raw_sequence,boolean,miairr,TRUE if single cells
    ↪  were isolated into separate compartments
Repertoire,sample.cell_number,protected,raw_sequence,integer,miairr,Total number of
    ↪ cells that went into the experiment
Repertoire,sample.cells_per_reaction,protected,raw_sequence,integer,miairr,Number of
    ↪ cells for each biological replicate
Repertoire,sample.cell_storage,protected,raw_sequence,boolean,miairr,TRUE if cells were
    ↪ cryo-preserved between isolation and further processing
Repertoire,sample.cell_quality,protected,raw_sequence,string,miairr,Relative amount of
    ↪ viable cells after preparation and (if applicable) thawing
Repertoire,sample.cell_isolation,protected,raw_sequence,string,miairr,Description of the
    ↪  procedure used for marker-based isolation or enrich cells
Repertoire,sample.cell_processing_protocol,protected,raw_sequence,string,miairr,
    ↪ Description of the methods applied to the sample including cell preparation/
    ↪ isolation/enrichment and nucleic acid extraction. This should closely mirror the
    ↪ Materials and methods section in the manuscript
Repertoire,sample.template_class,protected,raw_sequence,string,miairr,The class of
    ↪ nucleic acid that was used as primary starting material for the following
    ↪ procedures
Repertoire,sample.template_quality,protected,raw_sequence,string,miairr,Description and
    ↪ results of the quality control performed on the template material
```

```
Repertoire,sample.template_amount,protected,raw_sequence,string,miairr,Amount of
    ↪ template that went into the process
Repertoire,sample.library_generation_method,protected,raw_sequence,string,miairr,Generic
    ↪  type of library generation
Repertoire,sample.library_generation_protocol,protected,raw_sequence,string,miairr,
    ↪ Description of processes applied to substrate to obtain a library that is ready
    ↪ for sequencing
Repertoire,sample.library_generation_kit_version,protected,raw_sequence,string,miairr,"
    ↪ When using a library generation protocol from a commercial provider, provide the
    ↪ protocol version number"
Repertoire,sample.pcr_target.pcr_target_locus,protected,raw_sequence,string,miairr,"
    ↪ Designation of the target locus. Note that this field uses a controlled
    ↪ vocubulary that is meant to provide a generic classification of the locus, not
    ↪ necessarily the correct designation according to a specific nomenclature."
Repertoire,sample.pcr_target.forward_pcr_primer_target_location,protected,raw_sequence,
    ↪ string,miairr,Position of the most distal nucleotide templated by the forward
    ↪ primer or primer mix
Repertoire,sample.pcr_target.reverse_pcr_primer_target_location,protected,raw_sequence,
    ↪ string,miairr,Position of the most proximal nucleotide templated by the reverse
    ↪ primer or primer mix
Repertoire,sample.complete_sequences,protected,raw_sequence,string,miairr,"To be
    ↪ considered `complete`, the procedure used for library construction MUST generate
    ↪ sequences that 1) include the first V gene codon that encodes the mature
    ↪ polypeptide chain (i.e. after the leader sequence) and 2) include the last
    ↪ complete codon of the J gene (i.e. 1 bp 5' of the J->C splice site) and 3)
    ↪ provide sequence information for all positions between 1) and 2). To be
    ↪ considered `complete & untemplated`, the sections of the sequences defined in
    ↪ points 1) to 3) of the previous sentence MUST be untemplated, i.e. MUST NOT
    ↪ overlap with the primers used in library preparation."
Repertoire,sample.physical_linkage,protected,raw_sequence,string,miairr,Describes the
    ↪ mode of linkage if a method was used which physically links nucleic acids derived
    ↪  from distinct loci in a single-cell context.
Repertoire,sample.sequencing_run_id,protected,raw_sequence,string,miairr,ID of
    ↪ sequencing run assigned by the sequencing facility
Repertoire,sample.total_reads_passing_qc_filter,protected,raw_sequence,integer,miairr,
    ↪ Number of usable reads for analysis
Repertoire,sample.sequencing_platform,protected,raw_sequence,string,miairr,Designation
    ↪ of sequencing instrument used
Repertoire,sample.sequencing_facility,protected,raw_sequence,string,miairr,Name and
    ↪ address of sequencing facility
Repertoire,sample.sequencing_run_date,protected,raw_sequence,string,miairr,Date of
    ↪ sequencing run
Repertoire,sample.sequencing_kit,protected,raw_sequence,string,miairr,"Name,
    ↪ manufacturer, order and lot numbers of sequencing kit"
Repertoire,sample.sequencing_files.file_type,protected,raw_sequence,string,miairr,File
    ↪ format for the raw reads or sequences
Repertoire,sample.sequencing_files.filename,protected,raw_sequence,string,miairr,File
    ↪ name for the raw reads or sequences. The first file in paired-read sequencing
Repertoire,sample.sequencing_files.read_direction,protected,raw_sequence,string,miairr,
    ↪ Read direction for the raw reads or sequences. The first file in paired-read
    ↪ sequencing
Repertoire,sample.sequencing_files.read_length,protected,raw_sequence,integer,miairr,
    ↪ Read length in bases for the first file in paired-read sequencing
Repertoire,sample.sequencing_files.paired_filename,protected,raw_sequence,string,miairr,
    ↪ File name for the second file in paired-read sequencing
Repertoire,sample.sequencing_files.paired_read_direction,protected,raw_sequence,string,
    ↪ miairr,Read direction for the second file in paired-read sequencing
```

```
Repertoire,sample.sequencing_files.paired_read_length,protected,raw_sequence,integer,
    ↪ miairr,Read length in bases for the second file in paired-read sequencing
Repertoire,data_processing.data_processing_id,protected,statistics,string,airr-core,
    ↪ Identifier for the data processing object.
Repertoire,data_processing.primary_annotation,protected,statistics,boolean,airr-core,"If
    ↪  true, indicates this is the primary or default data processing for the
    ↪ repertoire and its rearrangments. If false, indicates this is a secondary or
    ↪ additional data processing."
Repertoire,data_processing.software_versions,protected,statistics,string,miairr,"Version
    ↪  number and/or date, include company pipelines"
Repertoire,data_processing.paired_reads_assembly,protected,statistics,string,miairr,How
    ↪ paired end reads were assembled into a single receptor sequence
Repertoire,data_processing.quality_thresholds,protected,statistics,string,miairr,How
    ↪ sequences were removed from (4) based on base quality scores
Repertoire,data_processing.primer_match_cutoffs,protected,statistics,string,miairr,"How
    ↪ primers were identified in the sequences, were they removed/masked/etc?"
Repertoire,data_processing.collapsing_method,protected,statistics,string,miairr,The
    ↪ method used for combining multiple sequences from (4) into a single sequence in
    ↪ (5)
Repertoire,data_processing.data_processing_protocols,protected,statistics,string,miairr,
    ↪ General description of how QC is performed
Repertoire,data_processing.data_processing_files,protected,statistics,array_string,airr-
    ↪ schema,Array of file names for data produced by this data processing.
Repertoire,data_processing.germline_database,protected,statistics,string,miairr,Source
    ↪ of germline V(D)J genes with version number or date accessed.
Repertoire,data_processing.analysis_provenance_id,protected,statistics,string,airr-
    ↪ schema,Identifier for machine-readable PROV model of analysis provenance
Rearrangement,sequence_id,protected,raw_sequence,string,airr-core,"Unique query sequence
    ↪  identifier within the file. Most often this will be the input sequence header or
    ↪  a substring thereof, but may also be a custom identifier defined by the tool in
    ↪ cases where query sequences have been combined in some fashion prior to alignment
    ↪ ."
Rearrangement,sequence,protected,raw_sequence,string,airr-core,"The query nucleotide
    ↪ sequence. Usually, this is the unmodified input sequence, which may be reverse
    ↪ complemented if necessary. In some cases, this field may contain consensus
    ↪ sequences or other types of collapsed input sequences if these steps are
    ↪ performed prior to alignment."
Rearrangement,sequence_aa,protected,raw_sequence,string,airr-schema,Amino acid
    ↪ translation of the query nucleotide sequence.
Rearrangement,rev_comp,protected,raw_sequence,boolean,airr-core,"True if the alignment
    ↪ is on the opposite strand (reverse complemented) with respect to the query
    ↪ sequence. If True then all output data, such as alignment coordinates and
    ↪ sequences, are based on the reverse complement of 'sequence'."
Rearrangement,productive,protected,raw_sequence,boolean,airr-core,True if the V(D)J
    ↪ sequence is predicted to be productive.
Rearrangement,vj_in_frame,protected,raw_sequence,boolean,airr-schema,True if the V and J
    ↪  gene alignments are in-frame.
Rearrangement,stop_codon,protected,raw_sequence,boolean,airr-schema,True if the aligned
    ↪ sequence contains a stop codon.
Rearrangement,complete_vdj,protected,raw_sequence,boolean,airr-schema,"True if the
    ↪ sequence alignment spans the entire V(D)J region. Meaning, sequence_alignment
    ↪ includes both the first V gene codon that encodes the mature polypeptide chain (i
    ↪ .e., after the leader sequence) and the last complete codon of the J gene (i.e.,
    ↪ before the J-C splice site). This does not require an absence of deletions within
    ↪  the internal FWR and CDR regions of the alignment."
Rearrangement,locus,protected,raw_sequence,string,airr-schema,"Gene locus (chain type).
    ↪ Note that this field uses a controlled vocabulary that is meant to provide a
```

```
    ↪ generic classification of the locus, not necessarily the correct designation
    ↪ according to a specific nomenclature."
Rearrangement,v_call,protected,raw_sequence,string,miairr,"V gene with allele. If
    ↪ referring to a known reference sequence in a database the relevant gene/allele
    ↪ nomenclature should be followed (e.g., IGHV4-59*01 if using IMGT/GENE-DB)."
Rearrangement,d_call,protected,raw_sequence,string,miairr,"First or only D gene with
    ↪ allele. If referring to a known reference sequence in a database the relevant
    ↪ gene/allele nomenclature should be followed (e.g., IGHD3-10*01 if using IMGT/GENE
    ↪ -DB)."
Rearrangement,d2_call,protected,raw_sequence,string,airr-schema,"Second D gene with
    ↪ allele. If referring to a known reference sequence in a database the relevant
    ↪ gene/allele nomenclature should be followed (e.g., IGHD3-10*01 if using IMGT/GENE
    ↪ -DB)."
Rearrangement,j_call,protected,raw_sequence,string,miairr,"J gene with allele. If
    ↪ referring to a known reference sequence in a database the relevant gene/allele
    ↪ nomenclature should be followed (e.g., IGHJ4*02 if using IMGT/GENE-DB)."
Rearrangement,c_call,protected,raw_sequence,string,miairr,"Constant region gene with
    ↪ allele. If referring to a known reference sequence in a database the relevant
    ↪ gene/allele nomenclature should be followed (e.g., IGHG1*01 if using IMGT/GENE-DB
    ↪ )."
Rearrangement,sequence_alignment,protected,raw_sequence,string,airr-core,"Aligned
    ↪ portion of query sequence, including any indel corrections or numbering spacers,
    ↪ such as IMGT-gaps. Typically, this will include only the V(D)J region, but that
    ↪ is not a requirement."
Rearrangement,sequence_alignment_aa,protected,raw_sequence,string,airr-schema,Amino acid
    ↪  translation of the aligned query sequence.
Rearrangement,germline_alignment,protected,raw_sequence,string,airr-core,"Assembled,
    ↪ aligned, full-length inferred germline sequence spanning the same region as the
    ↪ sequence_alignment field (typically the V(D)J region) and including the same set
    ↪ of corrections and spacers (if any)."
Rearrangement,germline_alignment_aa,protected,raw_sequence,string,airr-schema,Amino acid
    ↪  translation of the assembled germline sequence.
Rearrangement,junction,protected,raw_sequence,string,miairr,"Junction region nucleotide
    ↪ sequence, where the junction is defined as the CDR3 plus the two flanking
    ↪ conserved codons."
Rearrangement,junction_aa,protected,raw_sequence,string,miairr,Amino acid translation of
    ↪  the junction.
Rearrangement,np1,protected,raw_sequence,string,airr-schema,Nucleotide sequence of the
    ↪ combined N/P region between the V gene and first D gene alignment or between the
    ↪ V gene and J gene alignments.
Rearrangement,np1_aa,protected,raw_sequence,string,airr-schema,Amino acid translation of
    ↪  the np1 field.
Rearrangement,np2,protected,raw_sequence,string,airr-schema,Nucleotide sequence of the
    ↪ combined N/P region between either the first D gene and J gene alignments or the
    ↪ first D gene and second D gene alignments.
Rearrangement,np2_aa,protected,raw_sequence,string,airr-schema,Amino acid translation of
    ↪  the np2 field.
Rearrangement,np3,protected,raw_sequence,string,airr-schema,Nucleotide sequence of the
    ↪ combined N/P region between the second D gene and J gene alignments.
Rearrangement,np3_aa,protected,raw_sequence,string,airr-schema,Amino acid translation of
    ↪  the np3 field.
Rearrangement,cdr1,protected,raw_sequence,string,airr-schema,Nucleotide sequence of the
    ↪ aligned CDR1 region.
Rearrangement,cdr1_aa,protected,raw_sequence,string,airr-schema,Amino acid translation
    ↪ of the cdr1 field.
Rearrangement,cdr2,protected,raw_sequence,string,airr-schema,Nucleotide sequence of the
    ↪ aligned CDR2 region.
```

```
Rearrangement,cdr2_aa,protected,raw_sequence,string,airr-schema,Amino acid translation
    ↪ of the cdr2 field.
Rearrangement,cdr3,protected,raw_sequence,string,airr-schema,Nucleotide sequence of the
    ↪ aligned CDR3 region.
Rearrangement,cdr3_aa,protected,raw_sequence,string,airr-schema,Amino acid translation
    ↪ of the cdr3 field.
Rearrangement,fwr1,protected,raw_sequence,string,airr-schema,Nucleotide sequence of the
    ↪ aligned FWR1 region.
Rearrangement,fwr1_aa,protected,raw_sequence,string,airr-schema,Amino acid translation
    ↪ of the fwr1 field.
Rearrangement,fwr2,protected,raw_sequence,string,airr-schema,Nucleotide sequence of the
    ↪ aligned FWR2 region.
Rearrangement,fwr2_aa,protected,raw_sequence,string,airr-schema,Amino acid translation
    ↪ of the fwr2 field.
Rearrangement,fwr3,protected,raw_sequence,string,airr-schema,Nucleotide sequence of the
    ↪ aligned FWR3 region.
Rearrangement,fwr3_aa,protected,raw_sequence,string,airr-schema,Amino acid translation
    ↪ of the fwr3 field.
Rearrangement,fwr4,protected,raw_sequence,string,airr-schema,Nucleotide sequence of the
    ↪ aligned FWR4 region.
Rearrangement,fwr4_aa,protected,raw_sequence,string,airr-schema,Amino acid translation
    ↪ of the fwr4 field.
Rearrangement,v_score,protected,raw_sequence,number,airr-schema,Alignment score for the
    ↪ V gene.
Rearrangement,v_identity,protected,raw_sequence,number,airr-schema,Fractional identity
    ↪ for the V gene alignment.
Rearrangement,v_support,protected,raw_sequence,number,airr-schema,"V gene alignment E-
    ↪ value, p-value, likelihood, probability or other similar measure of support for
    ↪ the V gene assignment as defined by the alignment tool."
Rearrangement,v_cigar,protected,raw_sequence,string,airr-core,CIGAR string for the V
    ↪ gene alignment.
Rearrangement,d_score,protected,raw_sequence,number,airr-schema,Alignment score for the
    ↪ first or only D gene alignment.
Rearrangement,d_identity,protected,raw_sequence,number,airr-schema,Fractional identity
    ↪ for the first or only D gene alignment.
Rearrangement,d_support,protected,raw_sequence,number,airr-schema,"D gene alignment E-
    ↪ value, p-value, likelihood, probability or other similar measure of support for
    ↪ the first or only D gene as defined by the alignment tool."
Rearrangement,d_cigar,protected,raw_sequence,string,airr-core,CIGAR string for the first
    ↪  or only D gene alignment.
Rearrangement,d2_score,protected,raw_sequence,number,airr-schema,Alignment score for the
    ↪  second D gene alignment.
Rearrangement,d2_identity,protected,raw_sequence,number,airr-schema,Fractional identity
    ↪ for the second D gene alignment.
Rearrangement,d2_support,protected,raw_sequence,number,airr-schema,"D gene alignment E-
    ↪ value, p-value, likelihood, probability or other similar measure of support for
    ↪ the second D gene as defined by the alignment tool."
Rearrangement,d2_cigar,protected,raw_sequence,string,airr-schema,CIGAR string for the
    ↪ second D gene alignment.
Rearrangement,j_score,protected,raw_sequence,number,airr-schema,Alignment score for the
    ↪ J gene alignment.
Rearrangement,j_identity,protected,raw_sequence,number,airr-schema,Fractional identity
    ↪ for the J gene alignment.
Rearrangement,j_support,protected,raw_sequence,number,airr-schema,"J gene alignment E-
    ↪ value, p-value, likelihood, probability or other similar measure of support for
    ↪ the J gene assignment as defined by the alignment tool."
```

```
Rearrangement,j_cigar,protected,raw_sequence,string,airr-core,CIGAR string for the J
    ↪ gene alignment.
Rearrangement,c_score,protected,raw_sequence,number,airr-schema,Alignment score for the
    ↪ C gene alignment.
Rearrangement,c_identity,protected,raw_sequence,number,airr-schema,Fractional identity
    ↪ for the C gene alignment.
Rearrangement,c_support,protected,raw_sequence,number,airr-schema,"C gene alignment E-
    ↪ value, p-value, likelihood, probability or other similar measure of support for
    ↪ the C gene assignment as defined by the alignment tool."
Rearrangement,c_cigar,protected,raw_sequence,string,airr-schema,CIGAR string for the C
    ↪ gene alignment.
Rearrangement,v_sequence_start,protected,raw_sequence,integer,airr-schema,Start position
    ↪  of the V gene in the query sequence (1-based closed interval).
Rearrangement,v_sequence_end,protected,raw_sequence,integer,airr-schema,End position of
    ↪ the V gene in the query sequence (1-based closed interval).
Rearrangement,v_germline_start,protected,raw_sequence,integer,airr-schema,Alignment
    ↪ start position in the V gene reference sequence (1-based closed interval).
Rearrangement,v_germline_end,protected,raw_sequence,integer,airr-schema,Alignment end
    ↪ position in the V gene reference sequence (1-based closed interval).
Rearrangement,v_alignment_start,protected,raw_sequence,integer,airr-schema,Start
    ↪ position of the V gene alignment in both the sequence_alignment and
    ↪ germline_alignment fields (1-based closed interval).
Rearrangement,v_alignment_end,protected,raw_sequence,integer,airr-schema,End position of
    ↪  the V gene alignment in both the sequence_alignment and germline_alignment
    ↪ fields (1-based closed interval).
Rearrangement,d_sequence_start,protected,raw_sequence,integer,airr-schema,Start position
    ↪  of the first or only D gene in the query sequence. (1-based closed interval).
Rearrangement,d_sequence_end,protected,raw_sequence,integer,airr-schema,End position of
    ↪ the first or only D gene in the query sequence. (1-based closed interval).
Rearrangement,d_germline_start,protected,raw_sequence,integer,airr-schema,Alignment
    ↪ start position in the D gene reference sequence for the first or only D gene (1-
    ↪ based closed interval).
Rearrangement,d_germline_end,protected,raw_sequence,integer,airr-schema,Alignment end
    ↪ position in the D gene reference sequence for the first or only D gene (1-based
    ↪ closed interval).
Rearrangement,d_alignment_start,protected,raw_sequence,integer,airr-schema,Start
    ↪ position of the first or only D gene in both the sequence_alignment and
    ↪ germline_alignment fields (1-based closed interval).
Rearrangement,d_alignment_end,protected,raw_sequence,integer,airr-schema,End position of
    ↪  the first or only D gene in both the sequence_alignment and germline_alignment
    ↪ fields (1-based closed interval).
Rearrangement,d2_sequence_start,protected,raw_sequence,integer,airr-schema,Start
    ↪ position of the second D gene in the query sequence (1-based closed interval).
Rearrangement,d2_sequence_end,protected,raw_sequence,integer,airr-schema,End position of
    ↪  the second D gene in the query sequence (1-based closed interval).
Rearrangement,d2_germline_start,protected,raw_sequence,integer,airr-schema,Alignment
    ↪ start position in the second D gene reference sequence (1-based closed interval).
Rearrangement,d2_germline_end,protected,raw_sequence,integer,airr-schema,Alignment end
    ↪ position in the second D gene reference sequence (1-based closed interval).
Rearrangement,d2_alignment_start,protected,raw_sequence,integer,airr-schema,Start
    ↪ position of the second D gene alignment in both the sequence_alignment and
    ↪ germline_alignment fields (1-based closed interval).
Rearrangement,d2_alignment_end,protected,raw_sequence,integer,airr-schema,End position
    ↪ of the second D gene alignment in both the sequence_alignment and
    ↪ germline_alignment fields (1-based closed interval).
Rearrangement,j_sequence_start,protected,raw_sequence,integer,airr-schema,Start position
    ↪  of the J gene in the query sequence (1-based closed interval).
```

```
Rearrangement,j_sequence_end,protected,raw_sequence,integer,airr-schema,End position of
    ↪ the J gene in the query sequence (1-based closed interval).
Rearrangement,j_germline_start,protected,raw_sequence,integer,airr-schema,Alignment
    ↪ start position in the J gene reference sequence (1-based closed interval).
Rearrangement,j_germline_end,protected,raw_sequence,integer,airr-schema,Alignment end
    ↪ position in the J gene reference sequence (1-based closed interval).
Rearrangement,j_alignment_start,protected,raw_sequence,integer,airr-schema,Start
    ↪ position of the J gene alignment in both the sequence_alignment and
    ↪ germline_alignment fields (1-based closed interval).
Rearrangement,j_alignment_end,protected,raw_sequence,integer,airr-schema,End position of
    ↪  the J gene alignment in both the sequence_alignment and germline_alignment
    ↪ fields (1-based closed interval).
Rearrangement,cdr1_start,protected,raw_sequence,integer,airr-schema,CDR1 start position
    ↪ in the query sequence (1-based closed interval).
Rearrangement,cdr1_end,protected,raw_sequence,integer,airr-schema,CDR1 end position in
    ↪ the query sequence (1-based closed interval).
Rearrangement,cdr2_start,protected,raw_sequence,integer,airr-schema,CDR2 start position
    ↪ in the query sequence (1-based closed interval).
Rearrangement,cdr2_end,protected,raw_sequence,integer,airr-schema,CDR2 end position in
    ↪ the query sequence (1-based closed interval).
Rearrangement,cdr3_start,protected,raw_sequence,integer,airr-schema,CDR3 start position
    ↪ in the query sequence (1-based closed interval).
Rearrangement,cdr3_end,protected,raw_sequence,integer,airr-schema,CDR3 end position in
    ↪ the query sequence (1-based closed interval).
Rearrangement,fwr1_start,protected,raw_sequence,integer,airr-schema,FWR1 start position
    ↪ in the query sequence (1-based closed interval).
Rearrangement,fwr1_end,protected,raw_sequence,integer,airr-schema,FWR1 end position in
    ↪ the query sequence (1-based closed interval).
Rearrangement,fwr2_start,protected,raw_sequence,integer,airr-schema,FWR2 start position
    ↪ in the query sequence (1-based closed interval).
Rearrangement,fwr2_end,protected,raw_sequence,integer,airr-schema,FWR2 end position in
    ↪ the query sequence (1-based closed interval).
Rearrangement,fwr3_start,protected,raw_sequence,integer,airr-schema,FWR3 start position
    ↪ in the query sequence (1-based closed interval).
Rearrangement,fwr3_end,protected,raw_sequence,integer,airr-schema,FWR3 end position in
    ↪ the query sequence (1-based closed interval).
Rearrangement,fwr4_start,protected,raw_sequence,integer,airr-schema,FWR4 start position
    ↪ in the query sequence (1-based closed interval).
Rearrangement,fwr4_end,protected,raw_sequence,integer,airr-schema,FWR4 end position in
    ↪ the query sequence (1-based closed interval).
Rearrangement,v_sequence_alignment,protected,raw_sequence,string,airr-schema,"Aligned
    ↪ portion of query sequence assigned to the V gene, including any indel corrections
    ↪  or numbering spacers."
Rearrangement,v_sequence_alignment_aa,protected,raw_sequence,string,airr-schema,Amino
    ↪ acid translation of the v_sequence_alignment field.
Rearrangement,d_sequence_alignment,protected,raw_sequence,string,airr-schema,"Aligned
    ↪ portion of query sequence assigned to the first or only D gene, including any
    ↪ indel corrections or numbering spacers."
Rearrangement,d_sequence_alignment_aa,protected,raw_sequence,string,airr-schema,Amino
    ↪ acid translation of the d_sequence_alignment field.
Rearrangement,d2_sequence_alignment,protected,raw_sequence,string,airr-schema,"Aligned
    ↪ portion of query sequence assigned to the second D gene, including any indel
    ↪ corrections or numbering spacers."
Rearrangement,d2_sequence_alignment_aa,protected,raw_sequence,string,airr-schema,Amino
    ↪ acid translation of the d2_sequence_alignment field.
```

```
Rearrangement,j_sequence_alignment,protected,raw_sequence,string,airr-schema,"Aligned
    ↪ portion of query sequence assigned to the J gene, including any indel corrections
    ↪  or numbering spacers."
Rearrangement,j_sequence_alignment_aa,protected,raw_sequence,string,airr-schema,Amino
    ↪ acid translation of the j_sequence_alignment field.
Rearrangement,c_sequence_alignment,protected,raw_sequence,string,airr-schema,"Aligned
    ↪ portion of query sequence assigned to the constant region, including any indel
    ↪ corrections or numbering spacers."
Rearrangement,c_sequence_alignment_aa,protected,raw_sequence,string,airr-schema,Amino
    ↪ acid translation of the c_sequence_alignment field.
Rearrangement,v_germline_alignment,protected,raw_sequence,string,airr-schema,Aligned V
    ↪ gene germline sequence spanning the same region as the v_sequence_alignment field
    ↪  and including the same set of corrections and spacers (if any).
Rearrangement,v_germline_alignment_aa,protected,raw_sequence,string,airr-schema,Amino
    ↪ acid translation of the v_germline_alignment field.
Rearrangement,d_germline_alignment,protected,raw_sequence,string,airr-schema,Aligned D
    ↪ gene germline sequence spanning the same region as the d_sequence_alignment field
    ↪  and including the same set of corrections and spacers (if any).
Rearrangement,d_germline_alignment_aa,protected,raw_sequence,string,airr-schema,Amino
    ↪ acid translation of the d_germline_alignment field.
Rearrangement,d2_germline_alignment,protected,raw_sequence,string,airr-schema,Aligned D
    ↪ gene germline sequence spanning the same region as the d2_sequence_alignment
    ↪ field and including the same set of corrections and spacers (if any).
Rearrangement,d2_germline_alignment_aa,protected,raw_sequence,string,airr-schema,Amino
    ↪ acid translation of the d2_germline_alignment field.
Rearrangement,j_germline_alignment,protected,raw_sequence,string,airr-schema,Aligned J
    ↪ gene germline sequence spanning the same region as the j_sequence_alignment field
    ↪  and including the same set of corrections and spacers (if any).
Rearrangement,j_germline_alignment_aa,protected,raw_sequence,string,airr-schema,Amino
    ↪ acid translation of the j_germline_alignment field.
Rearrangement,c_germline_alignment,protected,raw_sequence,string,airr-schema,Aligned
    ↪ constant region germline sequence spanning the same region as the
    ↪ c_sequence_alignment field and including the same set of corrections and spacers
    ↪ (if any).
Rearrangement,c_germline_alignment_aa,protected,raw_sequence,string,airr-schema,Amino
    ↪ acid translation of the c_germline_aligment field.
Rearrangement,junction_length,protected,raw_sequence,integer,airr-schema,Number of
    ↪ nucleotides in the junction sequence.
Rearrangement,junction_aa_length,protected,raw_sequence,integer,airr-schema,Number of
    ↪ amino acids in the junction sequence.
Rearrangement,np1_length,protected,raw_sequence,integer,airr-schema,Number of
    ↪ nucleotides between the V gene and first D gene alignments or between the V gene
    ↪ and J gene alignments.
Rearrangement,np2_length,protected,raw_sequence,integer,airr-schema,Number of
    ↪ nucleotides between either the first D gene and J gene alignments or the first D
    ↪ gene and second D gene alignments.
Rearrangement,np3_length,protected,raw_sequence,integer,airr-schema,Number of
    ↪ nucleotides between the second D gene and J gene alignments.
Rearrangement,n1_length,protected,raw_sequence,integer,airr-schema,Number of untemplated
    ↪  nucleotides 5' of the first or only D gene alignment.
Rearrangement,n2_length,protected,raw_sequence,integer,airr-schema,Number of untemplated
    ↪  nucleotides 3' of the first or only D gene alignment.
Rearrangement,n3_length,protected,raw_sequence,integer,airr-schema,Number of untemplated
    ↪  nucleotides 3' of the second D gene alignment.
Rearrangement,p3v_length,protected,raw_sequence,integer,airr-schema,Number of
    ↪ palindromic nucleotides 3' of the V gene alignment.
```

```
Rearrangement,p5d_length,protected,raw_sequence,integer,airr-schema,Number of
    ↪ palindromic nucleotides 5' of the first or only D gene alignment.
Rearrangement,p3d_length,protected,raw_sequence,integer,airr-schema,Number of
    ↪ palindromic nucleotides 3' of the first or only D gene alignment.
Rearrangement,p5d2_length,protected,raw_sequence,integer,airr-schema,Number of
    ↪ palindromic nucleotides 5' of the second D gene alignment.
Rearrangement,p3d2_length,protected,raw_sequence,integer,airr-schema,Number of
    ↪ palindromic nucleotides 3' of the second D gene alignment.
Rearrangement,p5j_length,protected,raw_sequence,integer,airr-schema,Number of
    ↪ palindromic nucleotides 5' of the J gene alignment.
Rearrangement,consensus_count,protected,raw_sequence,integer,airr-schema,"Number of
    ↪ reads contributing to the (UMI) consensus for this sequence. For example, the sum
    ↪  of the number of reads for all UMIs that contribute to the query sequence."
Rearrangement,duplicate_count,protected,raw_sequence,integer,miairr,"Copy number or
    ↪ number of duplicate observations for the query sequence. For example, the number
    ↪ of UMIs sharing an identical sequence or the number of identical observations of
    ↪ this sequence absent UMIs."
Rearrangement,cell_id,protected,raw_sequence,string,miairr,Identifier defining the cell
    ↪ of origin for the query sequence.
Rearrangement,clone_id,protected,raw_sequence,string,airr-schema,Clonal cluster
    ↪ assignment for the query sequence.
Rearrangement,rearrangement_id,protected,raw_sequence,string,airr-schema,"Identifier for
    ↪  the Rearrangement object. May be identical to sequence_id, but will usually be a
    ↪  universally unique record locator for database applications."
Rearrangement,repertoire_id,protected,raw_sequence,string,airr-schema,Identifier to the
    ↪ associated repertoire in study metadata.
Rearrangement,sample_processing_id,protected,raw_sequence,string,airr-schema,Identifier
    ↪ to the sample processing object in the repertoire metadata for this rearrangement
    ↪ . If the repertoire has a single sample then this field may be empty or missing.
    ↪ If the repertoire has multiple samples then this field may be empty or missing if
    ↪  the sample cannot be differentiated or the relationship is not maintained by the
    ↪  data processing.
Rearrangement,data_processing_id,protected,raw_sequence,string,airr-schema,Identifier to
    ↪  the data processing object in the repertoire metadata for this rearrangement. If
    ↪  this field is empty than the primary data processing object is assumed.
Rearrangement,germline_database,protected,raw_sequence,string,airr-schema,Source of
    ↪ germline V(D)J genes with version number or date accessed.
```

Appendix C

ADC Middleware Server Questionaire

Here is presented the questionnaire that was sent to the experts: Brian Corrie, Scott Christley, and Christian Busse, including their responses for each question.

Each question is multiple choice where the user answered from 1 to 5, where 1 indicates that the user strongly disagrees and 5 that the user strongly agrees. For each question, there was additionally the possibility of providing text comments to support the answer.

1. **Is access control based on studies the right level of granularity?**
    - **Brian**: Answer - 4; Comments - "I think this is the most logical level of access granularity"
    - **Scott**: Answer - 5;
    - **Christian**: Answer - 4; Comments - "In general, yes. It might however sometimes be of interest to provide access to certain groups of subjects, e.g. healthy controls."
2. **For each repository there is a specific instance of the keycloak server running. Is this desirable?**
    - **Brian**: Answer - 4; Comments - "Yes, I think the repositories will want to/need to manage their own repository. A shared key cloak would be problematic from a data stewardship perspective."
    - **Scott**: Answer - 4;
    - **Christian**: Answer - 5;
3. **Is it useful to be able to edit the resource fields and their attributes like protection or "include_fields"?**
    - **Brian**: Answer - 4;

- **Scott**: Answer - 4;
- **Christian**: Answer - 5; Comments - "The usage of "include_fields" is unclear to me"

4. **Is editing the fields in the CSV file in Excel user-friendly?**
   - **Brian**: Answer - 2; Comments - "Probably OK for an administrator, assuming it doesn't change often."
   - **Scott**: Answer - 2;
   - **Christian**: Answer - 4;
5. **Is it useful to be able to define custom access levels (by way of "access_scope")?**
   - **Brian**: Answer - 4;
   - **Scott**: Answer - 4; Comments - "It lacks the ability to use different access levels for a field for different studies"
   - **Christian**: Answer - 5;
6. **Is it useful to be able to specify additional fields, which do not belong to the AIRR standard, for the resources?**
   - **Brian**: Answer - 4;
   - **Scott**: Answer - 5; Comments - "it's likely that private data fields are additional beyond the AIRR standard"
   - **Christian**: Answer - 5;
7. **Is the mechanism chosen to trigger synchronization (password protected endpoint) user friendly?**
   - **Brian**: Answer - 2; Comments - "An admin role/action, so probably fine."
   - **Scott**: Answer - 2; Comments - "some functionality isn't clear to me. If a study is removed from the repository, e.g. temporarily while some metadata is being correct, does keycloak lose all the information about which users have access to the study? That would not be desirable. I would also think in a production system that the cache could periodically auto-synchronize, or handle the synchronization only when requests come in."
   - **Christian**: Answer - 5;
8. **Do you perceive this mechanism to be secure?**
   - **Brian**: Answer - 4;
   - **Scott**: Answer - 3; Comments - "Unclear, as I'm not sure how the password is obtained, update, sent, etc. If it's being passed in the header in the clear, that's not so good."

- **Christian**: Answer - 2; Comments - "Depends on whether this password is used for only this purpose or corresponds with the one of the "real" admin account. Latter one is a no-go."

9. **Is this approach, to only synchronize after a batch update to the repository data has been made, appropriate for the repositories in use?**

   - **Brian**: Answer - 4; Comments - "In our experience, updates happen relatively rarely"
   - **Scott**: Answer - 2; Comments - "If this is meant to be a true cache, the cache should be able to manage itself, recognize cache incoherency and so forth without manual intervention"
   - **Christian**: Answer - 3; Comments - "Cannot judge."

10. **With this approach to synchronization all of the resources of a repository are assigned to only one keycloak resource owner (which cannot be updated afterwards). Is this feasible?**

    - **Brian**: Answer - 2; Comments - "Not really sure of the implications of this, but this seems limiting..."
    - **Scott**: Answer - 2; Comments - "Depends upon the repository. If a repository manages private data for multiple owners then it will need to give those owners control over just their studies."
    - **Christian**: Answer - 4; Comments - "Will work for small repo, but I see potential problems for hosted services."

11. **When a synchronization occurs, any study resources in keycloak that are not in the repository are deleted. Is this desirable?**

    - **Brian**: Answer - 3; Comments - "Not sure... Again, in our experience these repositories are pretty stable, with data added incrementally and relatively rarely. So if one removed a study from a repository, this is a major operation. So a sync and cleanup seems to make sense. But to have resources in keycloak disappear might be confusing... A way to mark them as "missing" and not "active" might be better???"
    - **Scott**: Answer - 1; Comments - "ok, so this answers my earlier question. Access to private study data is often formalized through a binding legal agreement between institutions on behalf of select individuals. Those access rights should be "permanent" until they are revoked, and they should be easy to audit to periodically insure compliance."
    - **Christian**: Answer - 5;

12. **Is the trade-off to store the rearrangement ID to repertoire ID mapping in a volatile cache worthy?**

- **Brian**: Answer - 3; Comments - "Not sure of the implication, in particular in terms of performance. If it helps performance then probably worth it..."
- **Scott**: Answer - 3; Comments - "Not sure I understand how this actually worked, or what that cache was doing. As repositories may hold billions of rearrangements, having it be scalable and fast is essential."
- **Christian**: Answer - 3; Comments - "Cannot judge."

13. **Is it useful to be able to filter the response resource fields based on the access level granted to a user?**

    - **Brian**: Answer - 3; Comments - "I think this could be quite valuable, allowing the data steward to control specific field level access. It gives fine grained control for fields in a repository (but assumes the same level for a field across all studies for the repository I think). This is potentially confusing for the consumer in that it is hard to tell if for a given study the field is missing or the user just doesn't have access to that field. Needs some further thought... 8-)"
    - **Scott**: Answer - 4; Comments - "It isn't clear to me how many of these fields require access control. One of the overriding principles of the AIRR Data Commons is that all possible metadata should be queryable so that studies can be found. You need to be able to find the study in order to request access to it."
    - **Christian**: Answer - 5; Comments - "The hierarchy of "access_scope" is unclear, i.e. where is it defined that "statistics" in included in "raw_sequences". Are only complete sub-/supersets allowed?"

14. **Is making 2 requests for most of the endpoints an acceptable trade-off for protecting them?**

    - **Brian**: Answer - 4; Comments - "Not sure of the implications of this, but I think this is an acceptable tradeoff..."
    - **Scott**: Answer - 4; Comments - "this is probably fine for the repertoire endpoint as the number of repertoires in a repository is only expected to be in the thousands, those queries can be performed very fast."
    - **Christian**: Answer - 5;

15. **Does the protection of the POST /v1/rearrangement endpoint seem performant enough to be usable, given that it is made twice?**

    - **Brian**: Answer - 4; Comments - "I think so, but will require some performance testing to confirm..."
    - **Scott**: Answer - 3; Comments - "This really depends upon the query. Some specialized queries are designed to be run quickly, while others will take much longer. There's on-going discussion in AIRR CRWG about how to handle "long running queries", having to do them twice is certainly not desirable."

- **Christian**: Answer - 3; Comments - "Cannot judge?"

16. **Is this approach of exposing public information: requesting public fields with "fields" or "facets" ADC parameters a good approach, given that a user needs to call the /v1/public_fields endpoint?**

    - **Brian**: Answer - 4;
    - **Scott**: Answer - 2; Comments - "It's outside of the AIRR standard and tools that operate against the ADC API may fail because they expect fields to be returned from the service when asked."
    - **Christian**: Answer - 4;

17. **Is modifying the "filters" parameter to enforce access control for facets a good approach?**

    - **Brian**: Answer - 4;
    - **Scott**: Answer - 4; Comments - "Seems a nice simple design that takes advantage of the query language"
    - **Christian**: Answer - 3; Comments - "Cannot judge."

18. **When a user specifies a field in the "filters" parameter for which his access level for some resources does not grant him access to that field, those resource are filtered out (even if they have public fields). Is this acceptable? This is done to avoid information leaks**

    - **Brian**: Answer - 2; Comments - "Not sure, I think this would require some further thought..."
    - **Scott**: Answer - 2; Comments - "Unclear how this done for facets as the counting is performed by the repository. And the concepts are a slightly mixed, access control on a field is different from access control on a study/repertoire. Restricting access control on a field doesn't necessarily imply that it limits access any studies. If you have access to a study but not to a particular field, can you query that study using that field?"
    - **Christian**: Answer - 5; Comments - "User needs to receive an information why and which resources were excluded. Otherwise ok."

19. **By default the "filters" operators "contains", "in", "exclude", "<", ">", "<=", ">=" are blacklisted and disabled to avoid timing attacks (can be enabled in the config). Is this acceptable?**

    - **Brian**: Answer - 3; Comments - "For now, yes, but would need to asses the risk of turning them on, as we would want them enabled in a production environment."

- **Scott**: Answer - 1; Comments - ""contains" and "in" are vital operators for well-described use cases for the ADC API. For example, searching for a CDR3 substring is essential and uses "contains". The "in" operator is essentially a compact "or" operator and used frequently to limit rearrangement queries to a limited set of repertoires."
- **Christian**: Answer - 5;

20. **The endpoints GET /v1, GET /v1/info and the added GET /v1/public_fields are publicly accessible. Is this a good idea?**

    - **Brian**: Answer - 4;
    - **Scott**: Answer - 5; Comments - "clients need to know the service is up and operational, they need the info to help them properly construct queries."
    - **Christian**: Answer - 5;

21. **Currently "format": "tsv" is not supported on POST endpoints. Is this feature necessary?**

    - **Brian**: Answer - 4; Comments - "Yes, this is necessary. I meant to ask why this was disabled and what the issues were..."
    - **Scott**: Answer - 5; Comments - "this is expected to be the predominant format for rearrangements as it is the AIRR TSV format used by all tools. There is not an AIRR approved JSON format for rearrangements."
    - **Christian**: Answer - 2;

22. **Is the minimal ADC compliance flexible enough to be easily implemented by non ADC compliant repositories (such as immuneDB)?**

    - **Brian**: Answer - 3; Comments - "My hope would be that it would not be necessary to make exceptions for non-compliant repositories. My understanding is that eventually ImmuneDB will be ADC compliant, and I think that should probably be the target for the implementation..."
    - **Scott**: Answer - 3; Comments - "I don't understand how this is minimal, the 4 endpoints are essentially all of the ADC API except for the simple status and /info endpoints."
    - **Christian**: Answer - 4;

23. **(Optional) Would you modify a non-compliant repository you're involved with to expose a minimal ADC API to be able to be protected by this solution? Instead of integrating protection in the repository itself**

    - **Brian**: Answer - 4; Comments - "It seems that utilizing the proxy approach is very powerful, you get a lot... The cost is being ADC API compliant, and although I am

probably biases, I also believe that being ADC compliant is a benefit to a repository. So if you can get a secure repository for free by becoming ADC API compliant, that is a win-win!!!"

- **Scott**: Answer - 1; Comments - "VDJServer is ADC API compliant"
- **Christian**: Answer - 5;

24. **Do you agree with the approach taken to the problem of access control for ADC APIs?**

    - **Brian**: Answer - 5; Comments - "My only 5 star response - I like the solution and don't give away 5 stars easily 8-)"
    - **Scott**: Answer - 2; Comments - "It likely needs to be more flexible to handle the separation of authority within institutions. I'm also not sure that the mechanism of control is appropriate for the rearrangement endpoint and how it will be used. Then again we don't have strong use cases to inform us..."
    - **Christian**: Answer - 4; Comments - "Looks good in general, will need testing on our side to make a final call."

25. **Do you agree with the architecture chosen for the solution? e. g. middleware server, keycloak, ID cache**

    - **Brian**: Answer - 4; Comments - "I think the one remaining issue to be sorted out is the process of integrating other IdPs and possibly decoupling the identity and user token from the UMA token that provides access control. "
    - **Scott**: Answer - 4; Comments - "keycloak is a bit generic, as VDJServer already incorporates authorization mechanism, I'd be more likely to utilize them directly, versus outsource it to an external tool."
    - **Christian**: Answer - 5;

26. **Do you agree with a middleware server based solution for protecting repositories, instead of integrating security directly in the repository?**

    - **Brian**: Answer - 5; Comments - "Yes, see above WRT win-win! This provides security for any ADC API repository. That is quite powerful!"
    - **Scott**: Answer - 4; Comments - "I think the middleware approach is a nice solution"
    - **Christian**: Answer - 4; Comments - "We will have to see how much "logic" the parsing/filtering in the middleware will require in the long run. We could run into situations, in which the middleware does not support functionality of the repo. But it clearly simplifies the situtation for "minimal" repositories."

27. **Do you perceive the overall solution to be secure?**

    - **Brian**: Answer - 4; Comments - "Without knowing the internal details, because it is standards based, this seems like an excellent approach..."

- **Scott**: Answer - 4;
- **Christian**: Answer - 4; Comments - "In general yes for the server side. Client side is less clear to me, but we would be a threat model for that."

28. **(Optional) Would you use this solution to protect an ADC API enabled repository you're involved with?**

    - **Brian**: Answer - 4; Comments - "That is the intent 8-)"
    - **Scott**: Answer - 1; Comments - "As this time VDJServer data repository only holds public data. If we expand to private data, it would likely be in the context of our analysis portal where we already maintain private data for users, and thus would utilize those existing control mechanisms."
    - **Christian**: Answer - 5;

29. **Do you have any overall observations or comments?**

    - **Brian**: Answer - "Nice work!"