Data access

Master of Applied IT – Semester 2

Date	02-04-2024
Version	0.1
Student name	Lars van den Brandt
Student number	434565
Teachers	Nico Kuijpers (1st) & Casper Schellekens (2nd)

Contents

Context

Research on data access for the RobotLab project involves understanding the specific needs and criteria for accessing the data generated by the RobotLab project. This research aims to determine who should have access to the data, what criteria should govern that access, and to what extent access can be controlled or restricted through the use of NOMAD Oasis.

In the context of the RobotLab project, various stakeholders, including scientific researchers, industrial partners, and potentially commercial entities, may require access to the data for different purposes. Understanding the needs and roles of these stakeholders is crucial in defining access policies and ensuring that data access aligns with project objectives and security considerations.

NOMAD Oasis, as a data management framework, plays a central role in facilitating data access within the RobotLab project. Researching NOMAD's capabilities in terms of access control mechanisms, user authentication, and data sharing features will provide insights into its compatibility with the project's data access needs.

Through interviews with stakeholders, literature reviews, and possibly case studies, the research will aim to gather information on the preferred access policies, user roles, and any regulatory or compliance considerations that may impact data access. This information will then be used to evaluate NOMAD's ability to meet these requirements and identify any gaps or areas for improvement.

By conducting thorough research on data access needs and NOMAD's capabilities, the project can make informed decisions regarding the implementation and customization of NOMAD Oasis to support effective data management within the RobotLab project. This approach ensures that data access policies are aligned with project goals.

Questions

1. Who should have access to the data generated by the RobotLab project?
2. What are the criteria for accessing the data? (e.g., role-based access, project-based access)
3. To what extend can access to the data be controlled or restricted trough NOMAD Oasis?
4. How does NOMAD address data interoperability (from various sources)?

Who should have access?

The **K**ey **P**erformance **I**ndicators (KPI's) contains requirements that the system must adhere to. This list contains one KPI involving accessibility:

Title	**KPI**
Accessibility	The accessibility to participate in the lab. This includes geographically dispersed execution of experiments, with results also capable of integration, partly through simulations. Additionally, the lab results will be available to various (research and industrial) parties (this is related to security).

Table 1: KPI NLGroeifonds BigChemistry

Business requirements[1]

The valorization team, in collaboration with partners, will devise an **I**ntellectual **P**roperty (IP) strategy rooted in the principle that the ownership of inventions is tied to the inventors and the knowledge generated. However, IP concerning the functioning of RobotLab within RobotLab IP B.V. will be shared.

A comprehensive clause on Intellectual Property will be integrated into the collaboration agreement among the parties. IP will be a standing item on the agenda of General Board meetings. Given the significance of knowledge sharing and innovation, attention will also be directed towards publication rights, open-access publishing, and making findings accessible to the scientific community.

In cases of strong commercial interest or potential for commercialization and patentability, parties will commit to initiating patent applications before publication. It is understood that each partner/organization retains ownership of its own Background IP contributed. Partners may utilize each other's background knowledge during the project without granting rights.

Results obtained during the project will belong to the participating organizations that have produced them. In instances where multiple parties have contributed to the results, joint IP will be addressed through separate agreements on rights. IP not essential to the functioning of RobotLab will only be documented after a positive novelty search and market study.

Market research will ascertain the distance from market introduction, required investments, commercialization strategy, led by the knowledge owner(s). RobotLab B.V. will establish agreements with participants and users regarding IP generated through RobotLab usage.

Following the completion of the first RobotLab prototype, scientific researchers and industrial R&D departments can access the infrastructure and knowledge for a fee. With each development phase, the facility will offer improved options, attracting more users. Initial focus will be on expanding contacts with R&D departments in industries like paint (AKZO, van der Wijhe), food (PepsiCo), and biotechnology (DSM).

[1] NLGroeifonds_BigChemistry_submitted

The RobotLab is expected to drive industrial R&D across personal care, food, and cosmetics. Various business model options for IP licensing will be explored, necessitating an innovative IP strategy from the outset:

- Generating revenue through licensing the RobotLab blueprint for local hubs.
- Offering subscriptions for access to digital infrastructure, datasets, and algorithms.

Criteria for access

When designing an access control model for collaborative environments like the RobotLab, requirements for the system must be defined. [1]

- **Multiple, dynamic user roles**
  The model should seamlessly accommodate multiple access roles of users, allowing access rights to be deduced from their respective roles.
- **Collaboration Rights**
  Implement read and write operations, the model should extend protection to all actions that could impact multiple users, ensuring collaboration rights are safeguarded.
- **Ease of Specification**
  Users should find it intuitive and straightforward to define access parameters.
- **Efficient Storage and Evaluation**
  Access definitions must be stored and evaluated efficiently to ensure optimal system performance.
- **Automation**
  Implementing access control within multi-user applications should be streamlined and user-friendly.

By addressing these requirements comprehensively, a robust access control model tailored to the needs of collaborative workspace of the RobotLab can be created.

Available policies

Access control policies can be grouped into three main classes: [2]

- **Discretionary (DAC)**
  (authorization-based) policies control access based on the identity of the requestor and on access rules stating what requestors are (or are not) allowed to do.
- **Mandatory (MAC)**
  policies control access based on mandated regulations de-termined by a central authority.
- **Role-based (RBAC)**
  policies control access depending on the roles that users have within the system and on rules stating what accesses are allowed to users in given roles.

Discretionary Policy

Discretionary policy is where a policy where access control is enforced based on the identity of the requestors based on rules on who can or cannot execute which actions on which resources.

The access matrix model [2]
The access matrix model is a framework for describing discretionary access control. The original model is called access matrix since the authorization state, meaning the authorizations holding at a given time in the system, is represented as a matrix. (figure 1)

When creating the access matrix, all users and all 'objects' (e.g. files or programs) have to be identified. Each user will be granted access to an object, this will determine what rights the user has on the specific object.

	Objects			
Users		Formula x	Text y	Video z
	RobotLab	*Read*	*Read*	*Own*
	PepsiCo	*Own*	*Read*	*Read*
	Coca-Cola		*Own*	*read*

Table 2: Example of access matrix (discretionary policy)

Enforcing discretionary policies can be challenging, especially in environments with a large number of users and resources. Managing access control lists (ACLs) for individual users and resources can become unwieldy and prone to errors, making it difficult to maintain a secure access control configuration over time.

Mandatory Policy

Mandatory security policies enforce access control based on regulations mandated by a central authority. It is used for protecting the data from insider threats. [3] The most commonly used form of mandatory policy is the multilevel security policy, which operates on the classifications of subjects and objects within the system:

- **Objects**
  Objects are passive entities storing information.
- **Subjects**
  Subjects are active entities, such as processes or programs, that request access to objects.

Multilevel policy [2]
In multilevel mandatory policies, each object and subject is assigned an access class. This access class comprises two main components:

- **Security level**
  The security level represents the sensitivity of the information stored in the object or the trustworthiness of the user accessing the system. Common security levels include Top Secret (TS), Secret (S), Confidential (C), and Unclassified (U), typically organized in a hierarchical order.
- **Set of categories**
  The set of categories defines the functional area associated with subjects and objects, providing better security classifications.

The dominance relationship between access classes is crucial in multilevel mandatory policies. This relationship determines the authorization levels for accessing objects based on the security level and categories of both the subjects and the objects.

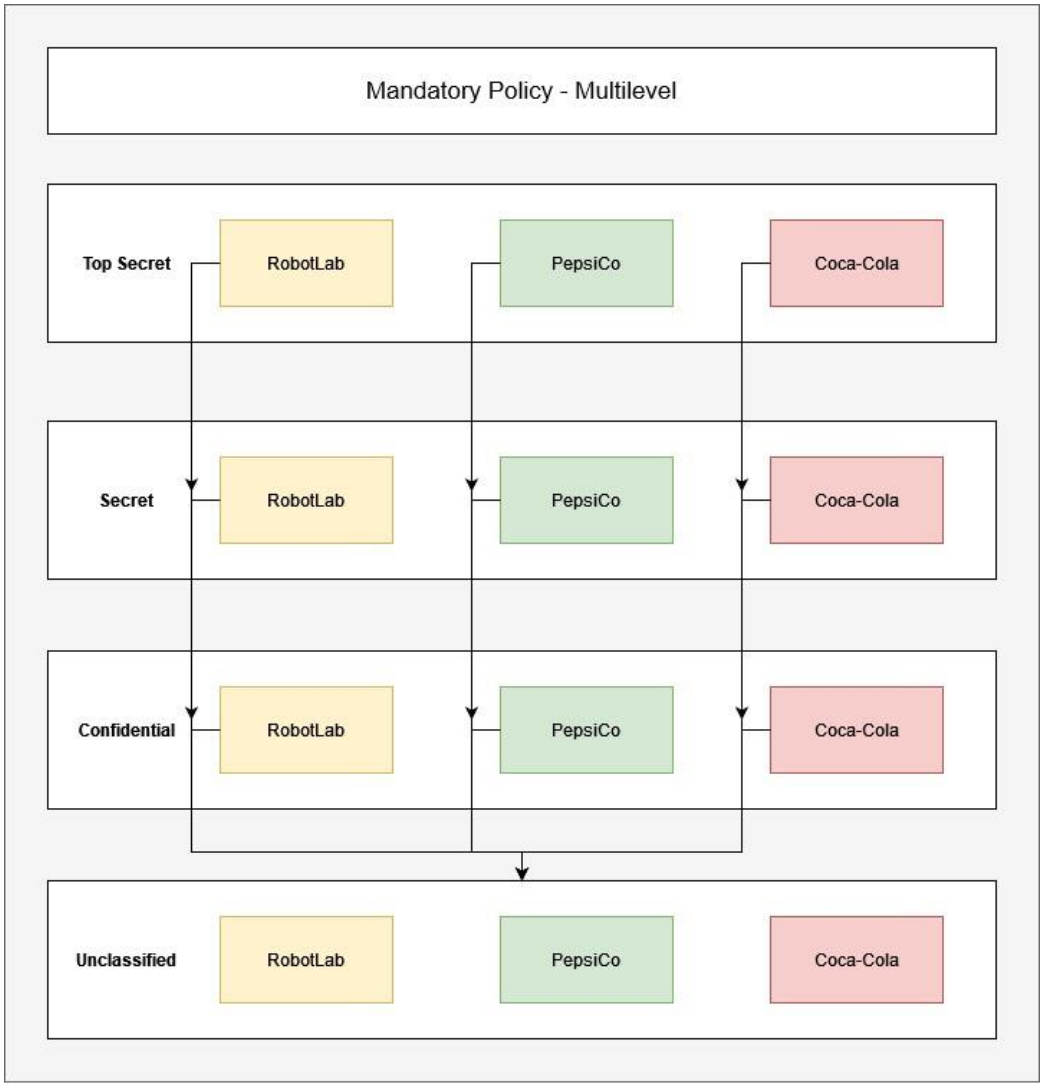

Figure 1: Mandatory Policy – Multilevel

While mandatory policies provide strong security guarantees, they may reduce flexibility in access control enforcement. Real-world scenarios often require exceptions to strict mandatory restrictions, such as data downgrading or waiving access restrictions in controlled situations. Therefore, a mandatory policy with multilevel systems should incorporate mechanisms to handle exceptions and allow controlled relaxation of restrictions when necessary.

Additionally, mandatory and discretionary policies can be applied simultaneously. This can be done by implementing discretionary policies within the boundaries defined by the security levels and categories of the mandatory policies, further enhancing access control capabilities.

Role-based Policy

Role-based access control is an increasingly popular in commercial applications. Role-based access control addresses the need to specify and enforce security policies that align naturally with an organization's structure. [2]

In Role-based access control, privileges are grouped into roles, each representing a set of actions and responsibilities associated with a particular organizational task or job function. Roles can be broadly scoped to reflect job titles or more specific to represent specific tasks within an organization. Users are then assigned roles, and access authorizations are specified for these roles instead of individual users. Users can activate one or more roles, enabling them to perform actions associated with those roles.

Roles can be organized hierarchically, allowing for role specialization and generalization. Authorization implications can propagate from general roles to specialized ones, simplifying authorization management and ensuring the least privilege principle. [1]

Role-based access control allows users to activate only the roles necessary to perform specific tasks, minimizing the risk of unauthorized access or misuse of privileges. Users do not need to exercise powerful roles until required, reducing the potential for accidental errors or security breaches.

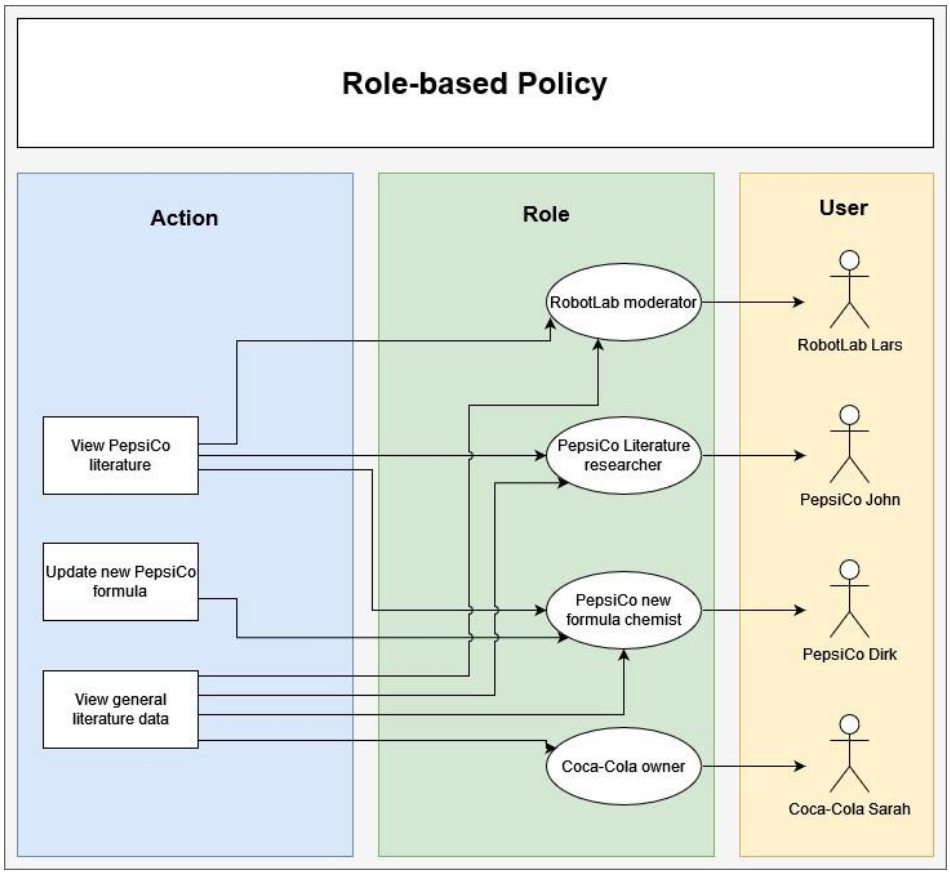

Figure 2: Role-based Policy

Despite its advantages, Role-based access control still has some limitations and areas for improvement. For example, the hierarchical relationship may not be sufficient to model all types of relationships that occur in real-world scenarios, and administrative policies may need to be enriched to accommodate various organizational structures and requirements.

Discussion on policies and models

The current access control policies, including discretionary, mandatory, and role-based, each present challenges that may not fully meet the specific requirements of the RobotLab project.

Discretionary Policy

The traditional access matrix model controls the access in the need-to-know like policy the RobotLab required. However, it can become hard to maintain in environments with a many different stakeholders, creating different users and objects. Managing individual access control lists (ACLs) for each user-object pair could lead to scalability issues and administrative overhead, particularly as the RobotLab project expands.

Mandatory Policy

While multilevel security policies offer robust protection, they may not align with the principle the need-to-know like policy the RobotLab requires. Assigning access based on security levels and categories may not adequately address the access requirements of various users and projects within the RobotLab.

Role-based Policy

While role-based access control offers a structured approach to access management, it may prove challenging to maintain in a dynamic environment like the RobotLab. With multiple involved parties, defining and managing roles could become hard to maintain and open to errors. Additionally, the assignment of roles may not meet the collaborative project needs, where researchers' involvement may vary over time.

Enhanced Access Matrix Model

By extending the discretionary policy with a new model, the access matrix can be innovated to manage access. In this enhanced model, objects explicitly specify the permissions granted to users, eliminating the need for maintaining complex user-object tables. This approach enhances clarity, simplifies administration, and improves the efficiency of access control within the RobotLab project.

In this model, each object contains 3 lists: Owner, Reader, Writer

	Roles			
Objects		Owner	Writer	Reader
	Formula x	RobotLab		
	Text y	PepsiCo	RobotLab	RobotLab
	Video z	PepsiCo		RobotLab, CocaCola

Table 3: Enhanced Access Matrix

In this model, each object (e.g., Formula x, Text y, Video z) explicitly lists the permissions granted to users. The available permissions include:

- **Read**
  Users can view the content of the object.
- **Write**
  Users can modify or add content to the object.
- **Own**
  Users have ownership rights over the object, can read, write and grant or revoke permission.
- **None**
  Users that are not mentioned in any of the above lists, have no permissions regarding the object.

By structuring the access matrix in this manner, owners of objects can easily manage access control by simply modifying the permissions associated with each object. This approach reduces complexity and enhances maintainability, particularly in large collaborative environments like the RobotLab project.

Furthermore, this model facilitates control over access rights, allowing administrators to tailor permissions to specific user roles or project requirements. For example, certain sensitive data may only be accessible to certain users with specific roles or responsibilities.

Overall, the enhanced access matrix model improves the efficiency and effectiveness of discretionary access control in collaborative environments, contributing to better security and streamlined management of resources.

NOMAD

Restricting access to the Oasis[2]

The Oasis operates similarly to the official NOMAD platform, allowing open access to published data. However, this might not align with your specific requirements.

Currently, there are two methods to completely restrict access to the Oasis.

Limiting Network Access

Restrict access to the Oasis by not exposing it to the public internet. Instead, it can be made available only on an intranet or through a VPN, ensuring that only authorized users can access it.

Whitelist Mechanism

Another option is to implement a white-list mechanism. As the Oasis administrator, a whitelist of allowed users can be configured as part of the Oasis settings. Users must be logged in and included in your whitelist to access the Oasis.

```
oasis:
    allowed_users:
        - user1@gmail.com
        - user2@gmail.com
```

Code 1: Whitelist users

Testing Restricting access

Testing the restriction of access to the Oasis completely involves verifying that at least one of the mechanisms controls this.

For the whitelist mechanism, testing involves several steps.

- A whitelist of allowed users is configured within the Oasis settings (code below).
- Restart the Oasis.
- Log in with allowed users.
- Log in with restricted users.

[2] https://nomad-lab.eu/prod/v1/staging/docs/howto/oasis/admin.html#restricting-access-to-your-oasis

```
oasis:
  is_oasis: true
  uses_central_user_management: true
    allowed_isers:
      - Lars.vandenbrandt@me.com
      - Lars.vandenbrandt@icloud.com
```

Code 2: Whitelist users

By testing, it's confirmed that the whitelist mechanism functions as intended, effectively restricting access to only authorized users. This security measure provides reassurance that unauthorized individuals cannot breach the Oasis's defences when the whitelist is implemented.

However, while the whitelist mechanism proves to be a valuable security enhancement, it falls short of meeting the specific data access management needs of the RobotLab project. Even though it restricts access to unauthorized users, once a user gets access, they have access to all data within the Oasis. This lack of data specific access control makes collaboration within the RobotLab project challenging, as it doesn't allow for the control over who can access specific data or resources.

Therefore, while the whitelist mechanism adds a layer of security, it does not provide the flexibility necessary to effectively manage data access within the project's scope. It is however a good additional security measurement.

Author rights[3]

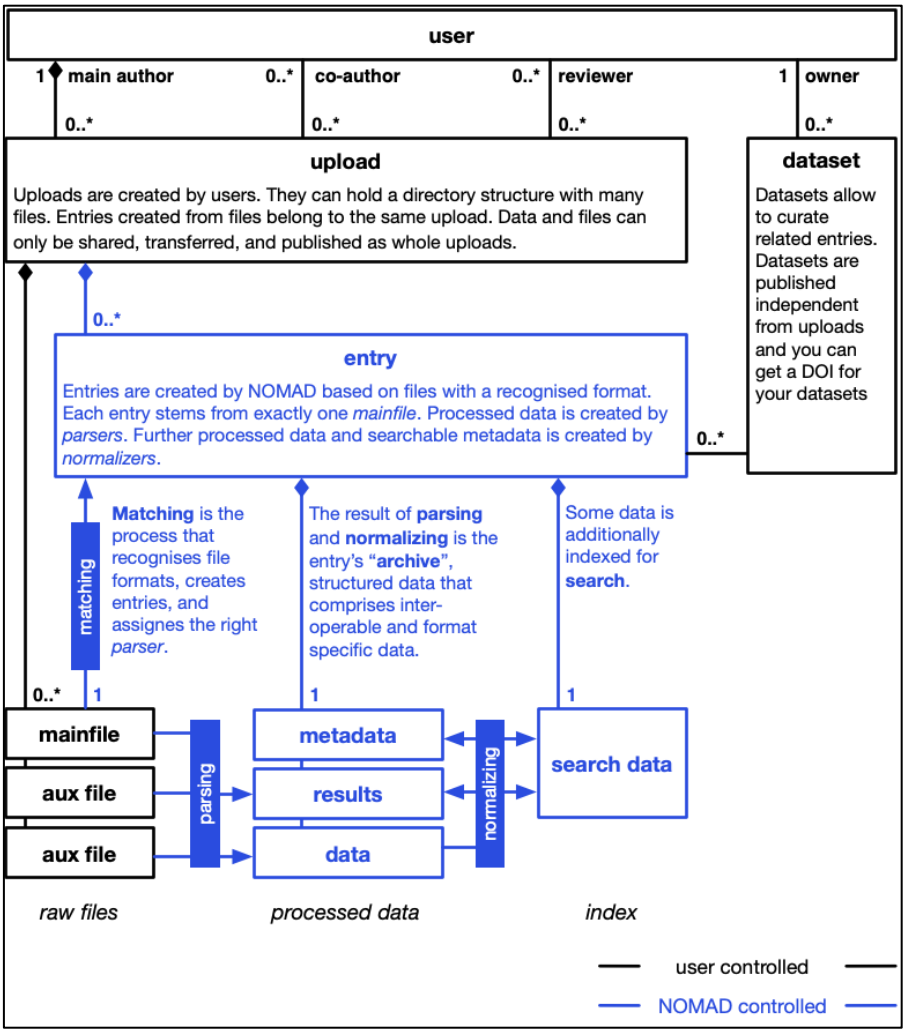

Figure 3: Data flow

Users create uploads to organize files. Think of an upload like a project or an experiment: many files can be put into a single upload and an upload can be structured with directories. Users can collaborate on uploads, share uploads, and publish uploads. The files in an upload are called raw files. Raw files are managed by users, and they are never changed by NOMAD. The raw files are saved in the regular file system.

Entries are the units of data processing in NOMAD. Each entry has a processed data or archive that is created from the mainfile by a parser. A parser is a small program that transforms data from a recognized mainfile into a structured machine processable tree of data. Only one parser is used for each entry. The used parser is determined during matching and depends on the file format. Parsers can be added to NOMAD as plugins.

[3] https://nomad-lab.eu/prod/v1/docs/explanation/basics.html#storing-and-indexing

An upload, whether published or unpublished, contains metadata.

```
"metadata": {
  "upload_create_time": "string",
  "publish_time": "string",
  "license": "string",
  "comment": "string",
  "entry_create_time": "string",
  "reviewers": "string",
  "external_id": "string",
  "coauthors": "string",
  "references": "string",
  "external_db": "string",
  "datasets": "string",
  "reviewer_groups": "string",
  "upload_name": "string",
  "embargo_length": 0,
  "coauthor_groups": "string",
  "main_author": "string"
}
```

Code 3: Upload metadata

This metadata, including, serves as vital information associated with the upload. Both published and unpublished uploads allow for the modification of their metadata. Within this context, author rights are mentioned. An author typically refers to a natural person who initially uploads a piece of data into NOMAD and holds authorship over it.

Within an upload, three distinct classes or roles exist for user access:

- **Main Author**
  This role is assigned to the individual who originally created the upload.
- **(Groups) Co-author**
  Co-authors possess the privilege to view and edit the content within an upload.
- **(Groups) Reviewers**
  Reviewers are granted access solely for viewing (or reviewing) an unpublished upload.

Before an upload is published in the Oasis, only users who are assigned one of these specified roles can access the upload and its content. However, once an upload is published in the Oasis, access extends to every user within the Oasis community. Nonetheless, only users with the appropriate roles retain the capability to edit some of the data post-publication.

Testing Author rights

User1 created an upload. When attempting to retrieve the metadata of a specific upload using User1's credentials, it shows the metadata response and indicates that no additional authors are associated with the upload. The response provided is as follows:


```
{
  "upload_id": "dhH0ZwkdQ5CrRUq5J3-EvA",
  "data": {
    "process_status": "SUCCESS",
    "complete_time": "2024-04-16T12:29:47.872000",
    "upload_id": "dhH0ZwkdQ5CrRUq5J3-EvA",
    "upload_name": "ApiTestUploadELN.zip",
    "upload_create_time": "2024-04-16T12:29:42.202000",
    "main_author": "1afae38d-59e8-4b67-aebb-72a5f0ce9ea8",
    "coauthors": [],
    "coauthor_groups": [],
    "reviewers": [],
    "reviewer_groups": [],
    "writers": [
      "1afae38d-59e8-4b67-aebb-72a5f0ce9ea8"
    ],
    "writer_groups": [],
    "viewers": [
      "1afae38d-59e8-4b67-aebb-72a5f0ce9ea8"
    ],
    "viewer_groups": [],
    "published": false,
    "published_to": [],
    "with_embargo": false,
    "embargo_length": 0,
    "license": "CC BY 4.0",
    "entries": 1,
  }
}
```


Code 4: Testing Author Rights: Response no authors added

Currently, there are no coauthors and reviewers added, and the upload is unpublished. User2, who should not have access to the upload, will get the following response:


```
{
  "detail": "You do not have access to the specified upload."
}
```


Code 5: Testing Author Rights: Response no access to upload

Currently, there are no coauthors and reviewers added, and the upload is unpublished. To further explore this functionality, a coauthor will be added to the upload, specifically the previously restricted User2.

Following this coauthor addition, an attempt will be made to retrieve the same upload using the credentials of User2 (who is now a coauthor):


```
{
  "upload_id": "dhH0ZwkdQ5CrRUq5J3-EvA",
  "data": {
    "process_running": false,
    "current_process": "edit_upload_metadata",
    "process_status": "SUCCESS",
    "complete_time": "2024-04-17T09:17:26.204000",
    "upload_id": "dhH0ZwkdQ5CrRUq5J3-EvA",
    "upload_name": "ApiTestUploadELN.zip",
    "upload_create_time": "2024-04-16T12:29:42.202000",
    "main_author": "1afae38d-59e8-4b67-aebb-72a5f0ce9ea8",
    "coauthors": [
      "2a61aaa6-16ba-4dfc-bb7d-9f730131502a"
    ],
    "coauthor_groups": [],
    "reviewers": [],
    "reviewer_groups": [],
    "writers": [
      "1afae38d-59e8-4b67-aebb-72a5f0ce9ea8",
      "2a61aaa6-16ba-4dfc-bb7d-9f730131502a"
    ],
    "writer_groups": [],
    "viewers": [
      "1afae38d-59e8-4b67-aebb-72a5f0ce9ea8",
      "2a61aaa6-16ba-4dfc-bb7d-9f730131502a"
    ],
    "viewer_groups": [],
    "published": false,
    "published_to": [],
    "with_embargo": false,
    "embargo_length": 0,
    "license": "CC BY 4.0",
    "entries": 1,
  }
}
```


Code 6: Testing Author Rights: Response added author

The second user now has access to the upload. Visible is the added coauthor.

While this method effectively establishes the necessary access control measures within the RobotLab, it's important to note its limitations. Primarily, this method only restricts access to uploads that are unpublished. Considering NOMAD's restriction of a maximum of 10 unpublished uploads per account, this approach does not address the requirements of data access management.

Enhanced Matrix Model

The enhanced access matrix can be implemented to manage access. In this enhanced model, objects explicitly specify the permissions granted to users, eliminating the need for maintaining complex user-object tables. This approach enhances clarity, simplifies administration, and improves the efficiency of access control within the RobotLab project.

The RobotLab produces data by researching by experimentation. Often, many experiments are involved for one research project. NOMAD supports this data architecture. An upload can be a research project, where each entry is one experiment.

For this test, there are 3 roles for each each upload. Each upload contains metadata which explicitly lists the permissions granted to users. The available roles and permissions include:

- **Reader**
  Users can view the content of the object.
- **Writer**
  Users can modify or add content to the object.
- **Owner**
  Users have ownership rights over the object, can read, write and grant or revoke permission.
- **None**
  Users that are not mentioned in any of the above lists, have no permissions regarding the object.

By structuring the access matrix in this manner, owners of objects can easily manage access control by simply modifying the permissions associated with each object. This this model facilitates control over access rights, allowing administrators to tailor permissions to specific users.

Testing enhanced matrix model with a script

By editing the queries which perform actions on uploads in a way that it will first check an access list in its metadata and checking if the current user's id is in the list, before approving the request, even when the upload is published will restrict access to data like the RobotLab requires.

For testing purposes, a proof of concept by a script will be created that runs as a service before NOMAD. These scripts will use the regular NOMAD Oasis openapi.json HTTP requests, but will add metadata with the defined roles to an upload.

This test will contain 3 actions.

- **Create Upload**
  Create an upload, and already define writers and readers by uploading a comment with defined roles.
- **Edit user access**
  As owner of an upload, edit the metadata of an upload. Ownership can be transferred/shared, and readers and writers can be added.
- **View Upload**
  If the upload is published, or unpublished, a user with an ID that is within one of the roles, has access to view the upload.

Create upload

- **Authentication Function** (`get_access_token`):
  - Retrieves an access token using username and password for basic authentication.
  - Constructs the token URL based on the provided NOMAD URL.
  - Makes a POST request to the token URL with authentication data.
  - Returns the access token if authentication is successful.
- **Upload Function** (`upload_file_with_metadata`):
  - Accepts username, password, file path, readers, and writers as inputs.
  - Calls the `get_access_token` function to obtain an access token.
  - Constructs headers with authorization using the access token.
  - Opens the file to be uploaded in binary mode.
  - Makes a POST request to upload the file to the NOMAD system.
  - Waits for the upload process to complete by checking the upload status.
  - Updates the metadata of the uploaded file with information about readers and writers.
  - Makes a POST request to edit the metadata of the uploaded file.
  - Returns True if the upload and metadata update are successful, False otherwise.
- **Main Function** (`main`):
  - Continuously prompts the user for username, password, file path, readers, and writers.
  - Calls the `upload_file_with_metadata` function with provided inputs.
  - Asks the user if they want to make another upload or try again in case of failure.

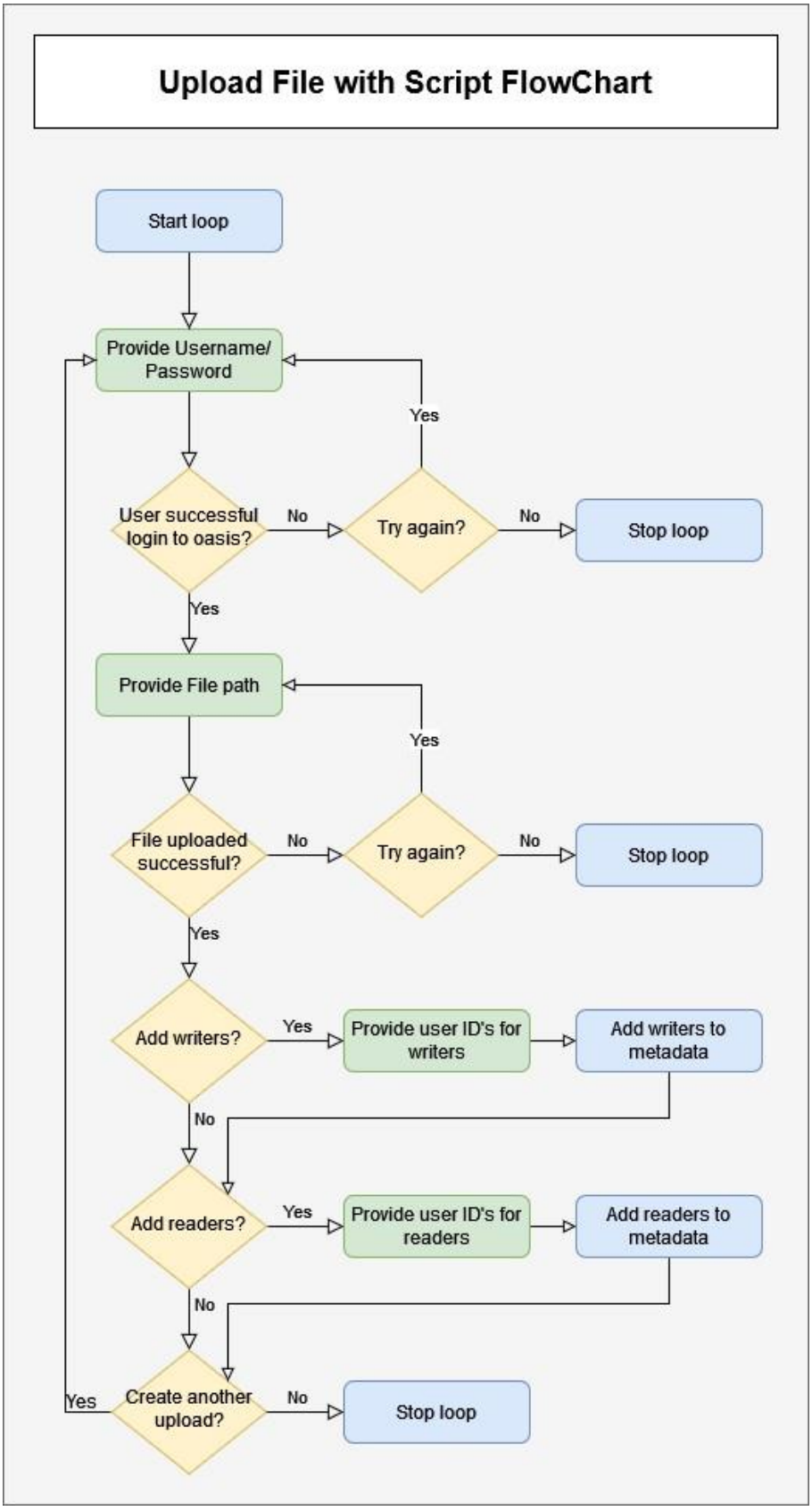

Figure 4: Upload script flowchart

Edit user access

- **Authentication Function** (`authenticate`):
  - Retrieves an access token using username and password for basic authentication.
  - Constructs the token URL based on the provided Keycloak URL.
  - Makes a POST request to the token URL with authentication data.
  - Returns the access token if authentication is successful.
- **Function to Get Upload Entries** (`get_upload_entries`):
  - Retrieves the entries of a specific upload.
  - Constructs the URL for fetching upload entries.
  - Makes a GET request to the URL with appropriate headers.
  - Returns the JSON response containing upload entries.
- **Function to Update Metadata** (`update_metadata`):
  - Updates metadata of the upload with a new comment.
  - Constructs the URL for editing metadata.
  - Extracts readers and writers from the comment.
  - Constructs payload and headers for the POST request.
  - Makes a POST request to update metadata.
  - Returns True if metadata update is successful, False otherwise.
- **Function to Get User Data** (`get_user_data`):
  - Retrieves the user data of the authenticated user.
  - Constructs the URL for fetching user data.
  - Makes a GET request to the URL with appropriate headers.
  - Returns the JSON response containing user data.
- **Functions for Adding and Removing Users from Roles:**
  - `add_user_to_role`: Adds a user ID to the specified role in the comment metadata.
  - `remove_user_from_role`: Removes a user ID from the specified role in the comment metadata.
  - `is_user_owner`: Checks if the user is one of the owners of the document.
- **Main Function** (`main`):
  - Continuously prompts the user for username, password, and upload ID.
  - Authenticates the user and retrieves access token.
  - Gets user data and upload entries.
  - Iterates through each entry, extracts comment and user ID, and checks if the user is the owner.
  - If the user is the owner, prompts for action (add/remove) and role (Owner/Writer/Reader).
  - Updates the metadata with the new comment after adding or removing the user.
  - Asks the user for further actions or retries based on the success/failure of metadata update.

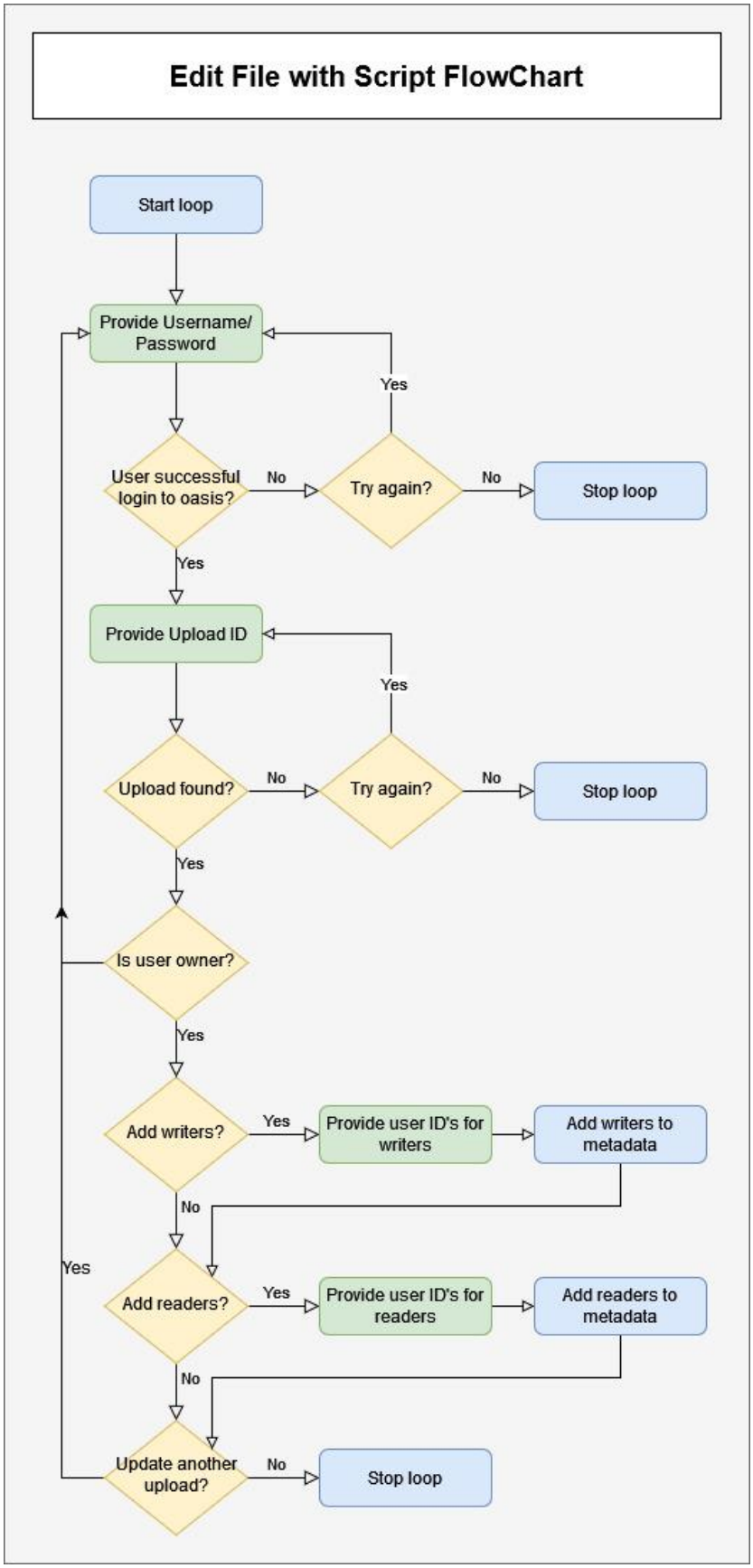

Figure 5: Edit upload script flowchart

View upload

- **Authentication Function** (`authenticate`):
  - Authenticates the user and retrieves the access token using username and password.
  - Constructs the token URL based on the provided Keycloak URL.
  - Makes a POST request to the token URL with authentication data.
  - Returns the access token if authentication is successful.
- **Functions to Retrieve Upload Entries** (`get_upload_entries`) **and User Data** (`get_user_data`):
  - Retrieve upload entries and user data of the authenticated user using the access token.
  - Construct appropriate URLs and headers for making HTTP requests.
  - Return JSON responses containing upload entries and user data.
- **Function to Parse Comment** (`parse_comment`):
  - Parses the comment string into a dictionary format.
  - Uses regular expressions to extract roles and corresponding user IDs.
  - Returns a dictionary with roles as keys and lists of user IDs as values.
- **Function to Check Permission** (`has_permission`):
  - Checks if the user has the specified role in the comment data.
  - Takes the parsed comment data, user ID, and role as inputs.
  - Returns True if the user has the role, False otherwise.
- **Main Function** (`main`):
  - Continuously prompts the user for username, password, and upload ID.
  - Authenticates the user and retrieves the access token.
  - Gets user data and upload entries.
  - Parses the comment for each entry and checks the user's permissions.
  - Prints the corresponding data if the user has permission or informs the user otherwise.
  - Asks the user if they want to check another upload or not.

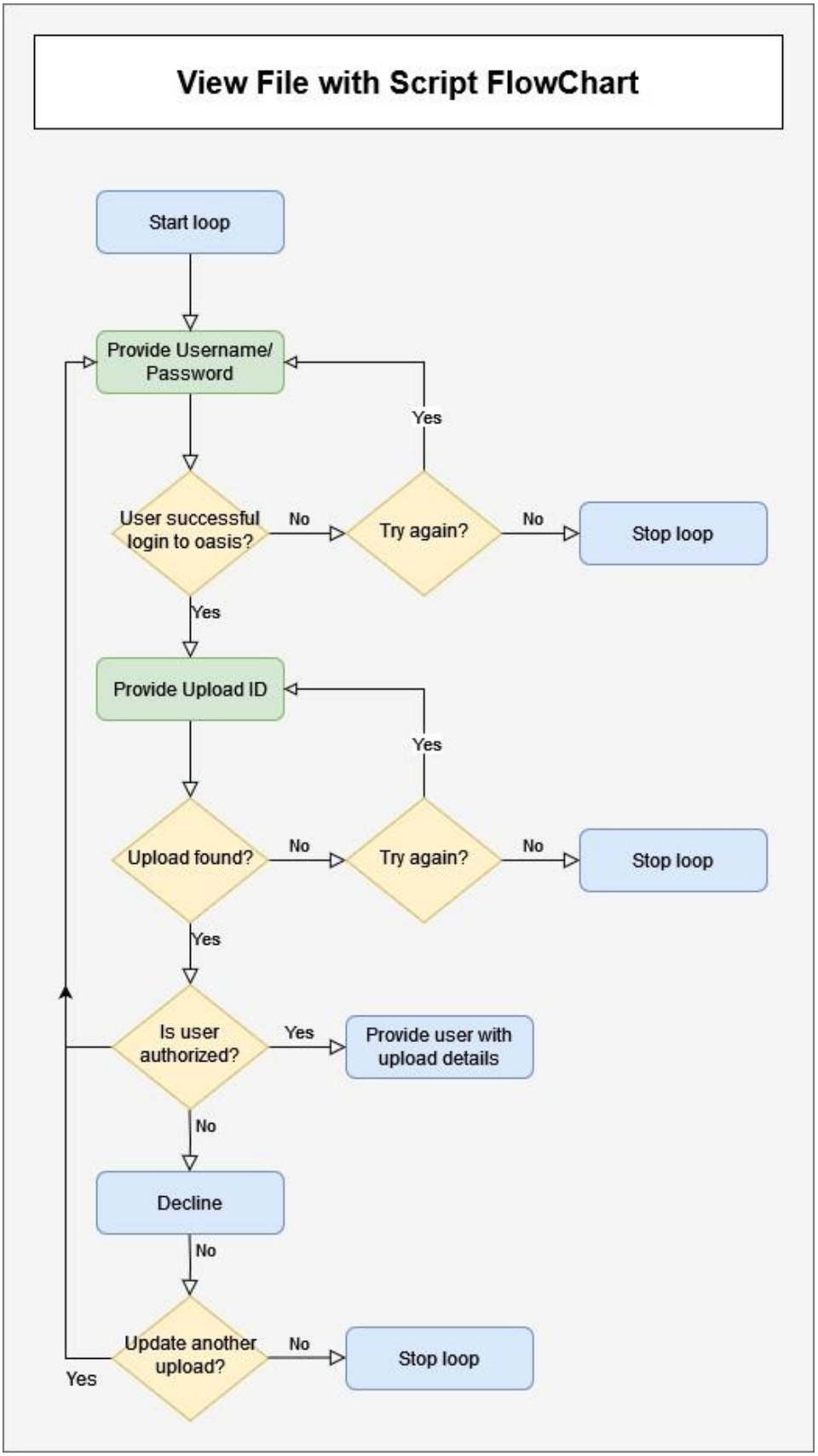

Figure 6: View upload script flowchart

Testing enhanced matrix model with code edit

NOMAD is Open-Source with Apache License Version 2.0, which means the code can be pulled and edited freely.

As mentioned previously, NOMAD uses reviewers and coauthors to check the authorization before an upload is published.

The original code, contains 2 functions that are used within the upload queries.

- **Is_user_upload_viewer**
  Returns if user is allowed to read a specific upload.
- **Is_user_upload_writer**
  Returns if user is allowed to write to a specific upload.

Changing the code

Original code:

```
def is_user_upload_viewer(upload: Upload, user: Optional[User]):
    if 'all' in upload.reviewer_groups:
        return True

    if user is None:
        return False

    if user.is_admin:
        return True

    if user.user_id in upload.viewers:
        return True

    group_ids = get_group_ids(user.user_id)
    if not set(group_ids).isdisjoint(upload.viewer_groups):
        return True

    return False


def is_user_upload_writer(upload: Upload, user: User):
    if user.is_admin:
        return True

    if user.user_id in upload.writers:
        return True

    group_ids = get_group_ids(user.user_id)
    if not set(group_ids).isdisjoint(upload.writer_groups):
        return True

    return False
```

Code 7: Original NOMAD code

By changing the validation to a check where only admins, reviewers and writers can view the code, the upload cannot be viewed by users who do not obtain any of these roles, even when the upload is published.

Edited code:

```
def is_user_upload_viewer(upload: Upload, user: Optional[User]):
    if user is None:
        return False

    if user.is_admin:
        return True

    if user.user_id in upload.writers:
        return True

    if user.user_id in upload.reviewers:
        return True

    return False


def is_user_upload_writer(upload: Upload, user: User):
    if user.is_admin:
        return True

    if user.user_id in upload.writers:
        return True

    group_ids = get_group_ids(user.user_id)
    if not set(group_ids).isdisjoint(upload.writer_groups):
        return True

    return False
```

Code 8: Updated NOMAD code

An example of a function where this is used is the get_upload_with_read_access function, where the edited is_user_upload_viewer is called. It is then checked if the user has the right permissions and responds accordingly.

```
def get_upload_with_read_access(
    upload_id: str, user: Optional[User], include_others: bool = False
) -> Upload:
    """
    Determines if the user has read access to the upload. If so, the
corresponding Upload object is returned.
    """
    mongodb_query = _query_mongodb(upload_id=upload_id)
    upload = mongodb_query.first()
    if upload is None:
        raise HTTPException(
            status_code=status.HTTP_404_NOT_FOUND,
            detail=strip(
                """
            The specified upload_id was not found."""
            ),
        )

    if is_user_upload_viewer(upload, user):
        return upload

    if not include_others:
        raise HTTPException(
            status_code=status.HTTP_401_UNAUTHORIZED,
            detail=strip(
                """
            You do not have access to the specified upload."""
            ),
        )
```

Code 9: NOMAD get upload function

Testing code changes

Because the source code of the Oasis is edited, access is controlled on both the GUI and the openAPI. For demonstration purposes, an upload is created through the openAPI.

Creating an upload

An upload is created with Account1.

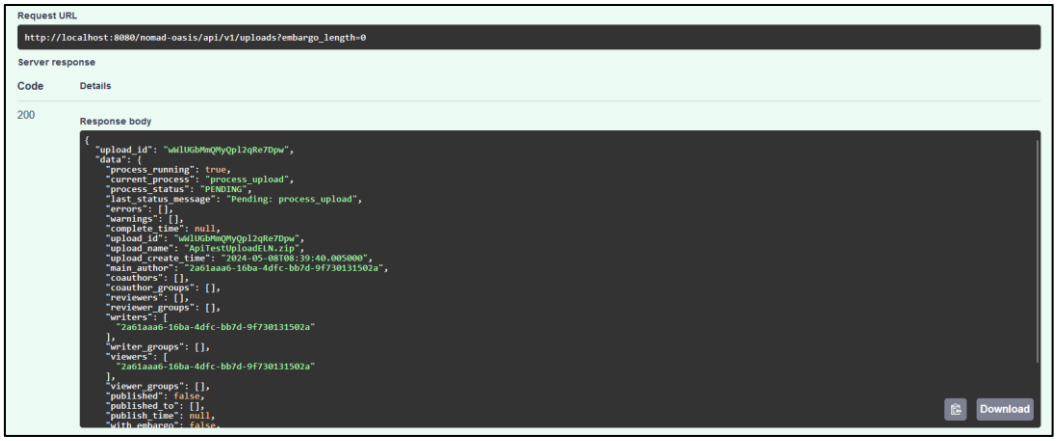

Figure 7: Create an upload OpenAPI

Viewing the upload

Currently the upload can be viewed by the uploader, Account1 (main author), and the Oasis admin.

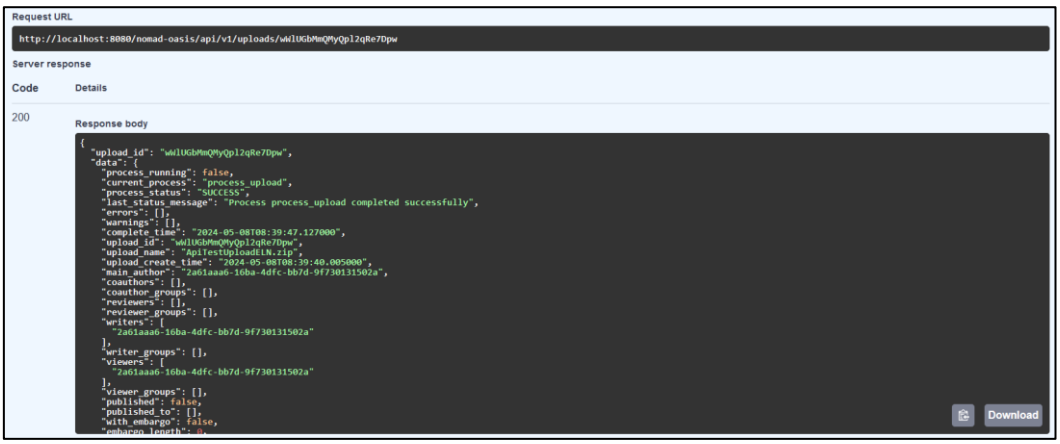

Figure 8: Viewing an upload acc1 OpenAPI

But trying to view the upload with Account2, will give access denied message.

Figure 9: Viewing an upload acc2 denied OpenAPI

Changing user rights to upload

The user_id of Account2 is added to the reviewers list of the upload.

Request body:

```
{
  "metadata": {
    "reviewers": ["1afae38d-59e8-4b67-aebb-72a5f0ce9ea8"]
  }
}
```

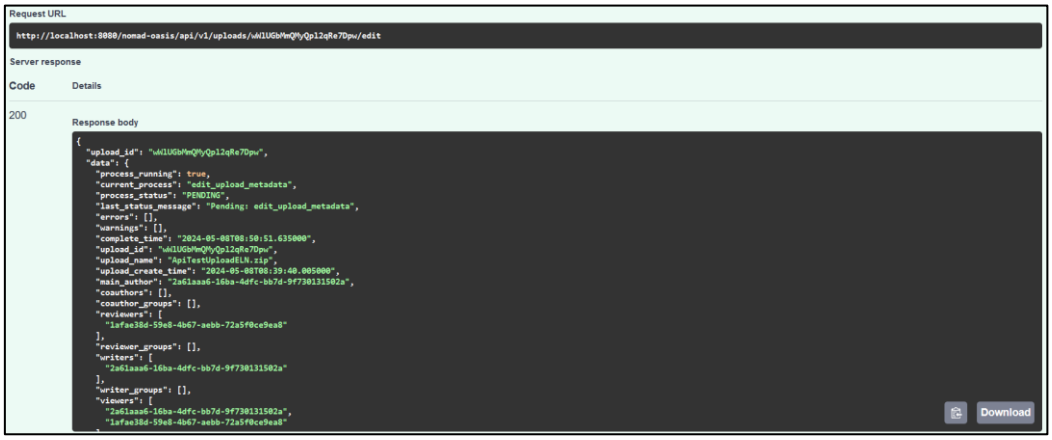

Figure 10: Changing user rights OpenAPI

Now, Account2 can view the upload.

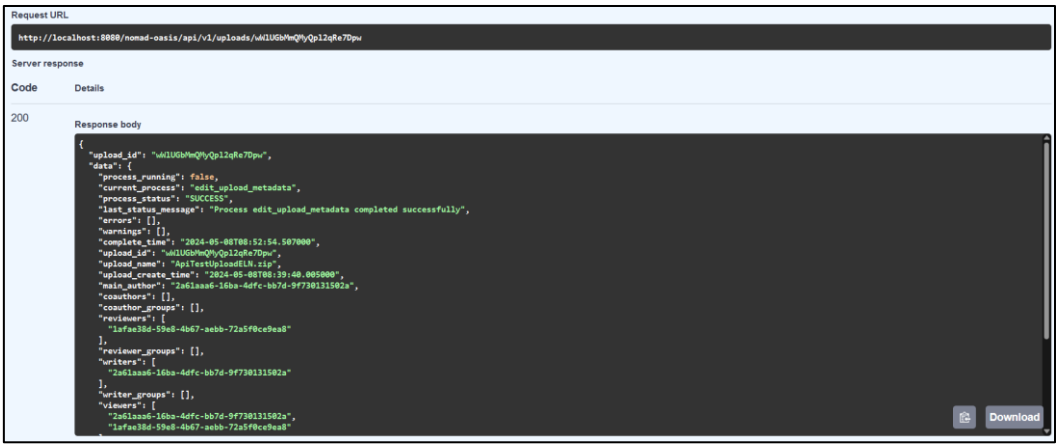

Figure 11: Viewing an upload acc2 allowed OpenAPI

Conclusion

In conclusion, the RobotLab project requires an enhanced matrix model to effectively manage access to its data due to the involvement of various stakeholders with varying roles and responsibilities, with possible collaboration between them. While the standard NOMAD Oasis platform offers the capability to restrict access to the entire Oasis through whitelisting mechanisms, it falls short of providing the control over access to specific data resources for the RobotLab project.

Fortunately, NOMAD's open-source nature under the Apache License Version 2.0 allows for the modification of its source code to tailor it to specific needs. Adjustments can be made to NOMAD's codebase to extend access control functionalities beyond the pre-publishing phase.

NOMAD already incorporates roles such as main authors, co-authors, and reviewers to regulate access before an upload is published. By modifying the code to enforce these roles even after publication, the access policy can be effectively implemented. This ensures that only authorized users, including main authors, co-authors, and reviewers, can access specific data within the Oasis, aligning with the requirements of the RobotLab project.

In essence, by enhancing NOMAD's access control mechanisms through code modifications, the RobotLab project can achieve the necessary level of data access control, enabling seamless collaboration while safeguarding sensitive information.

Bibliography

[1] H. Shen and P. Dewan, "Access control for collaborative environments," 1992.

[2] P. Samarati and S. D. C. D. Vimercati, "Access Control: Policies, Models, and Mechanisms," *Lecture Notes in Computer Science,* pp. 137-196, 2001.

[3] E. Bertino, G. Ghinita and A. Kamra, "Access Control for Databases: Concepts and Systems," *Foundations and Trends in Databases,* vol. 2, no. 1-2, pp. 1-148, 2010.

Code snippets

Figures

Tables