

# Graph-based Provenance Metadata Database Documentation

---

Hochschule Karlsruhe  
University of  
Applied Sciences



Summer semester 2024

Developed by Philip Mader

Supervised by José Antonio Peregrina Pérez & Prof. Dr. rer. nat. Christian Zirpins

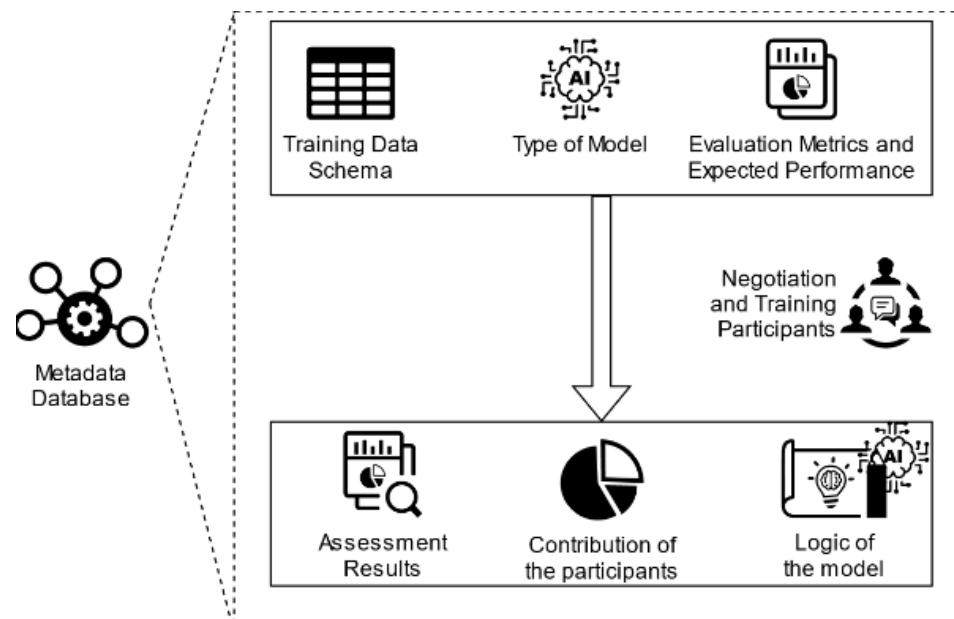
## Table of Contents

---

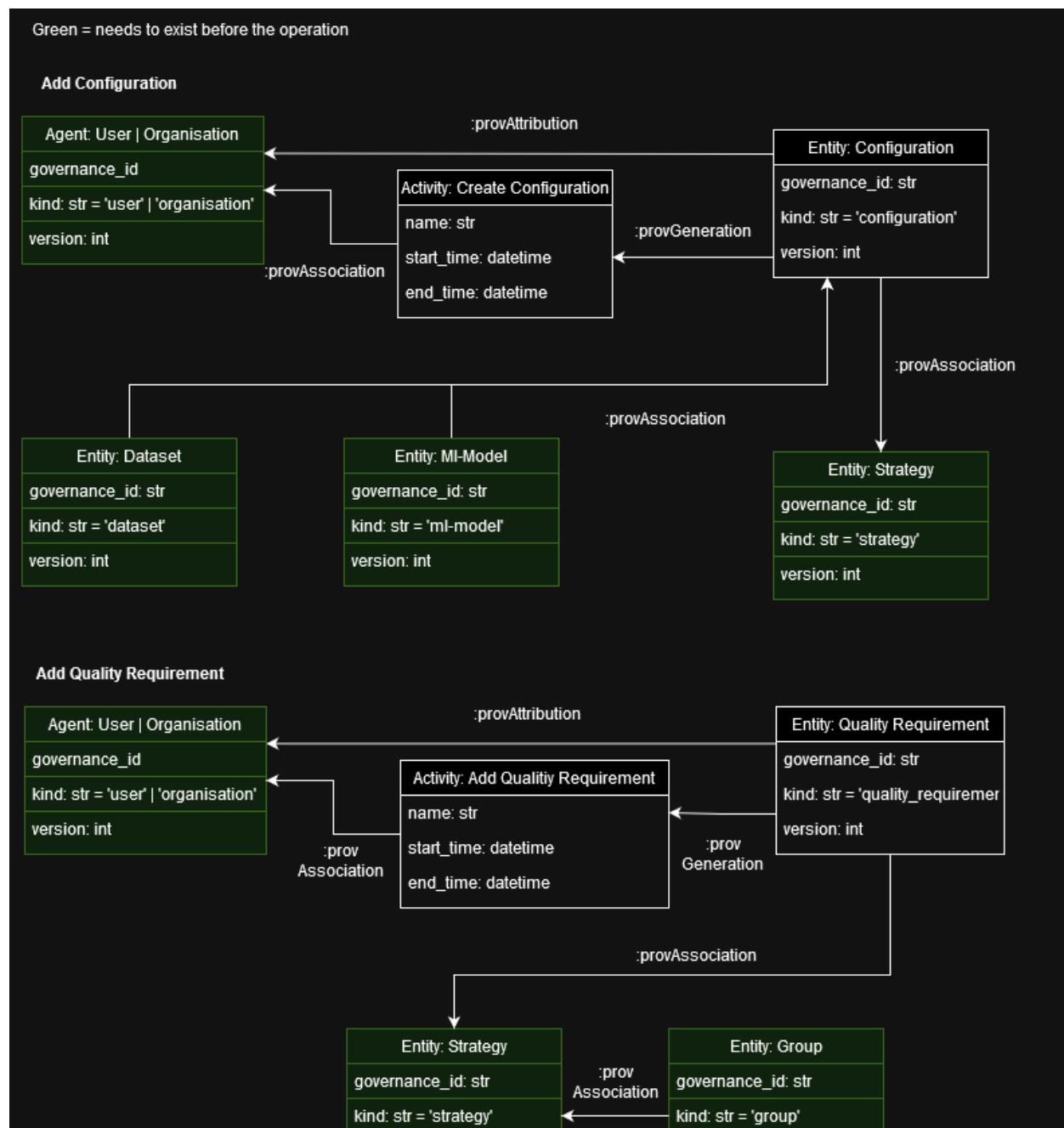
- [Introduction](#)
- [Requirements](#)
- [Usage](#)
- [Metadata Middleware](#)
  - [Description](#)
  - [Requirements](#)
  - [Usage](#)
  - [Operations](#)
- [Neo4JConnection](#)
  - [Description](#)
  - [Requirements](#)
  - [Usage](#)
  - [Operations](#)
- [Metadata-API](#)
  - [Description](#)
  - [Requirements](#)
  - [Usage](#)
  - [Operations](#)

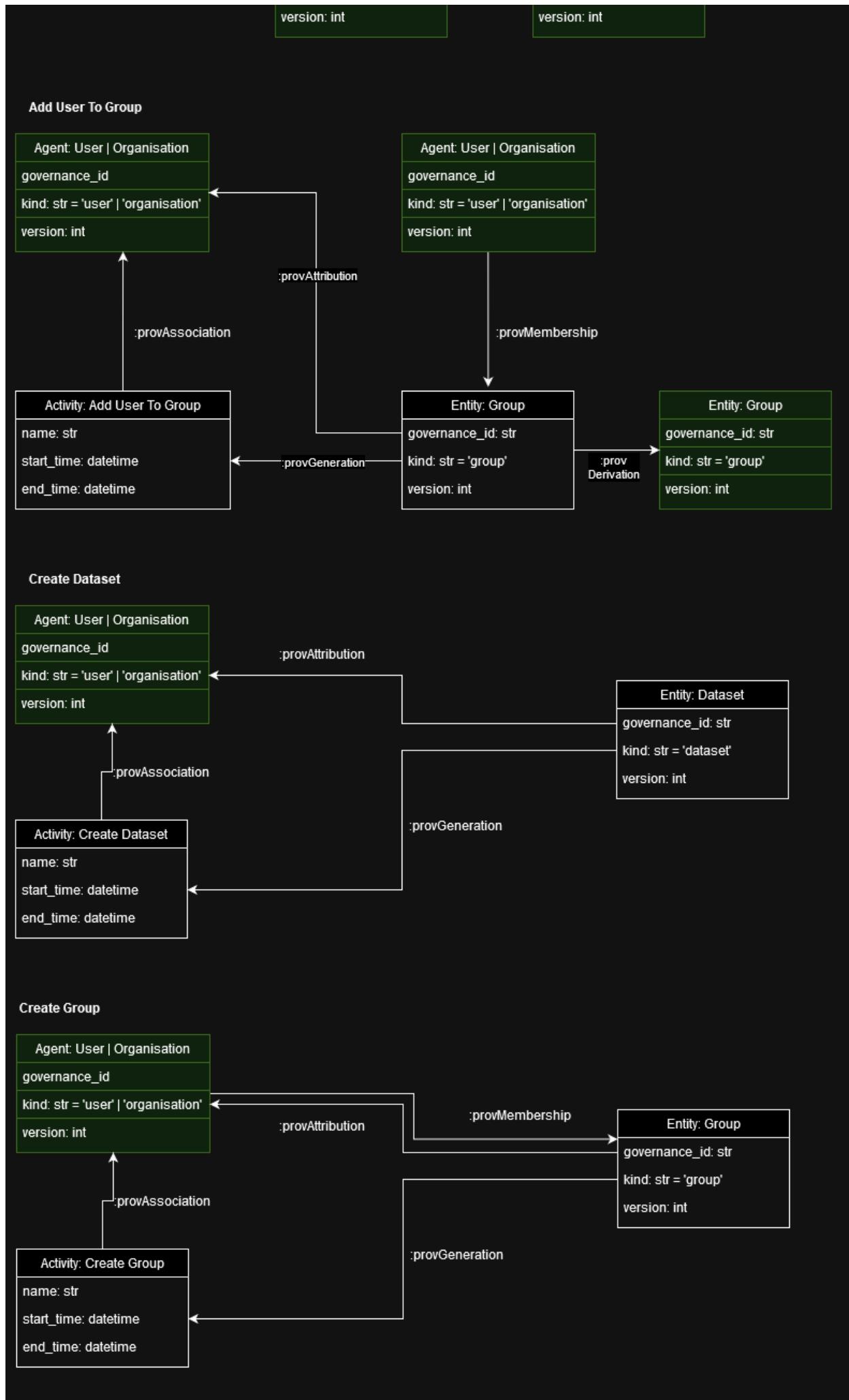
## Introduction

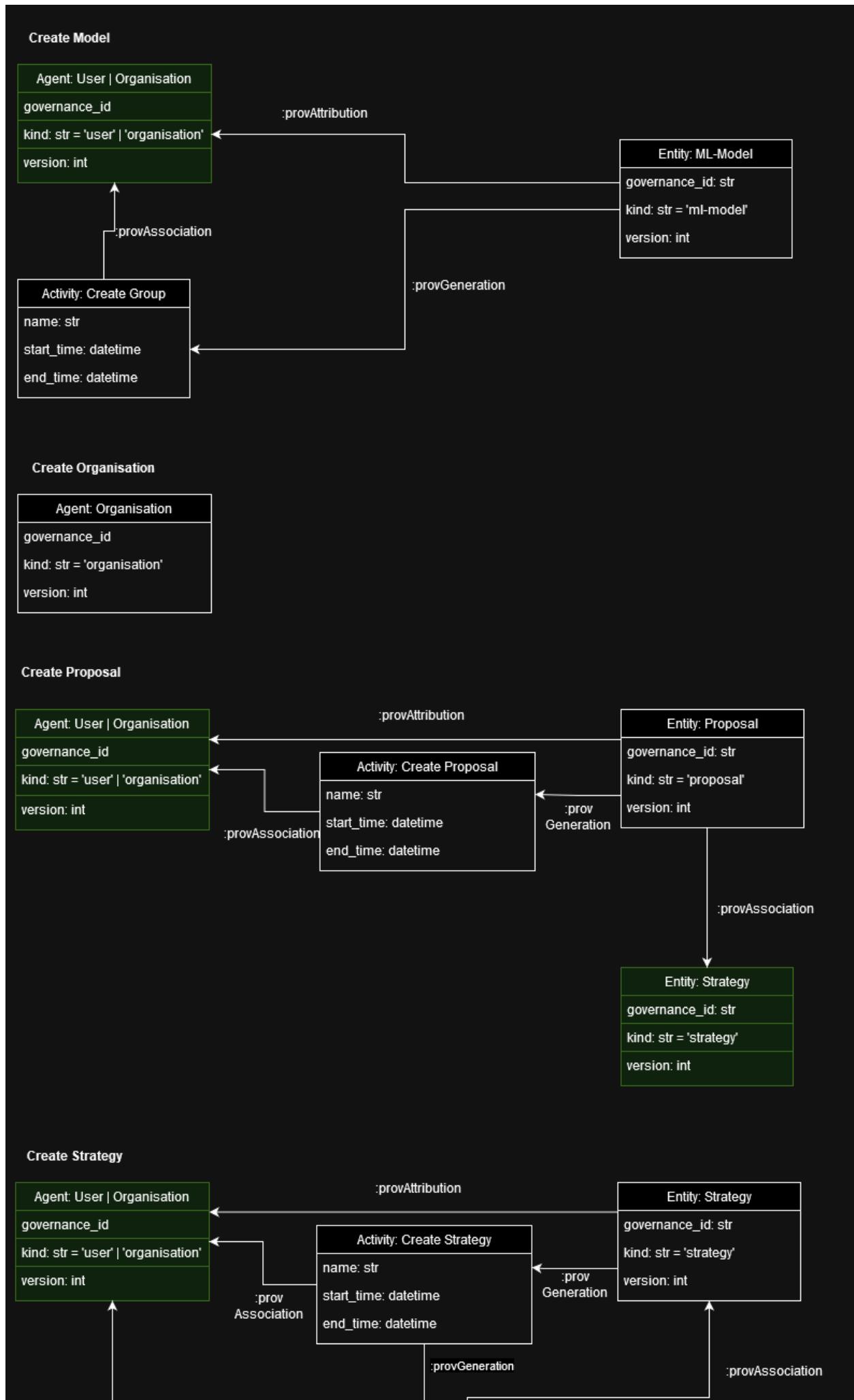
The goal of this project is to manage provenance metadata within the Aura project. In this case provenance metadata refers to a record of operations taken by both data providers and engineers in the [Data Governance Cockpit](#). This data can then be used to create a link between the operations and evaluations of the model created by the federated learning process. This way if, for example, the evaluation results worsen significantly there is a database containing a record of all actions that can be queried in convenient ways to find the cause of the bad results.

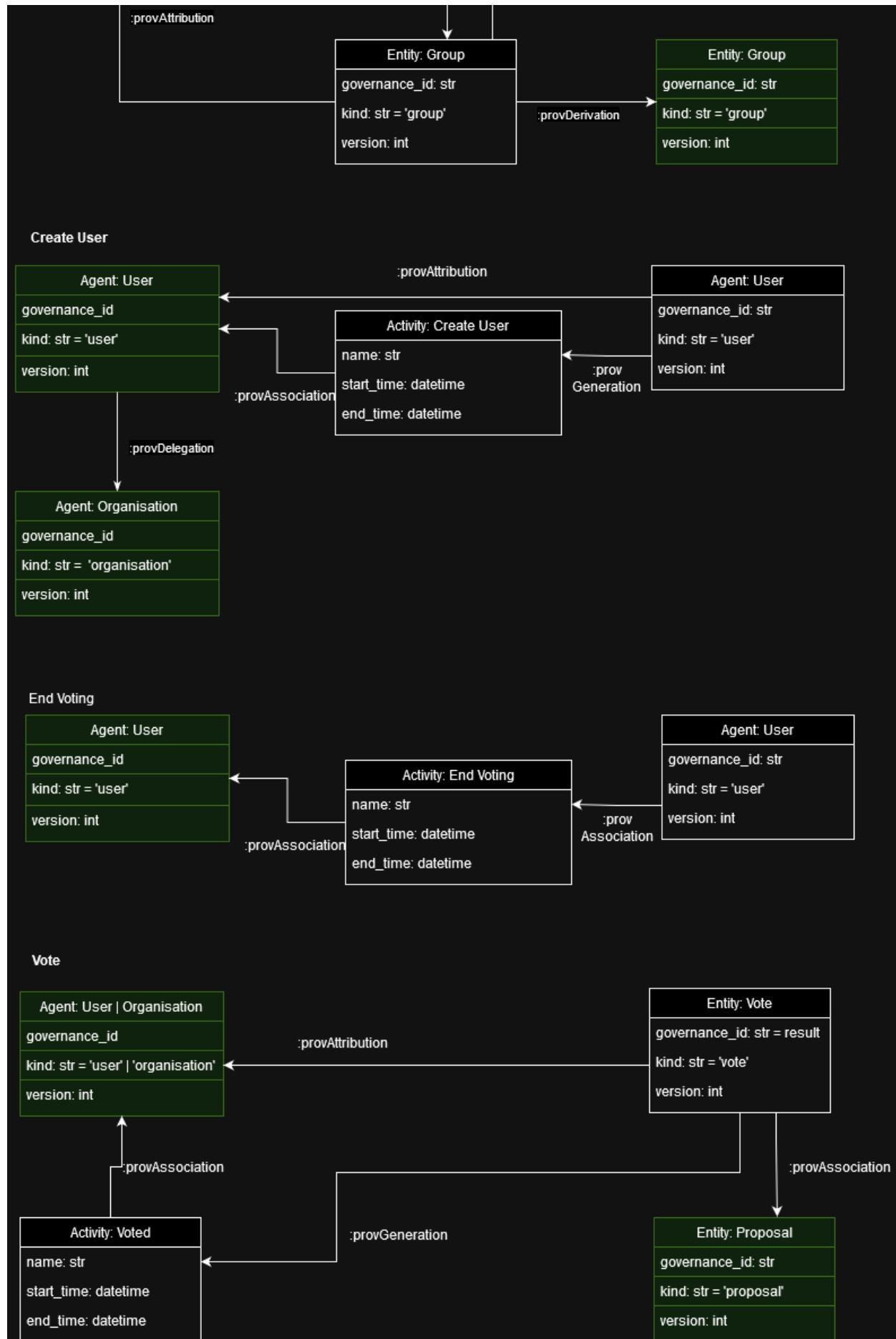


To accomplish this the software within this package generates provenance metadata from the operations in the [Data Governance Cockpit](#) and saves it in an instance of Neo4j. Specifically the newly created database entries for each write operation are as follows:







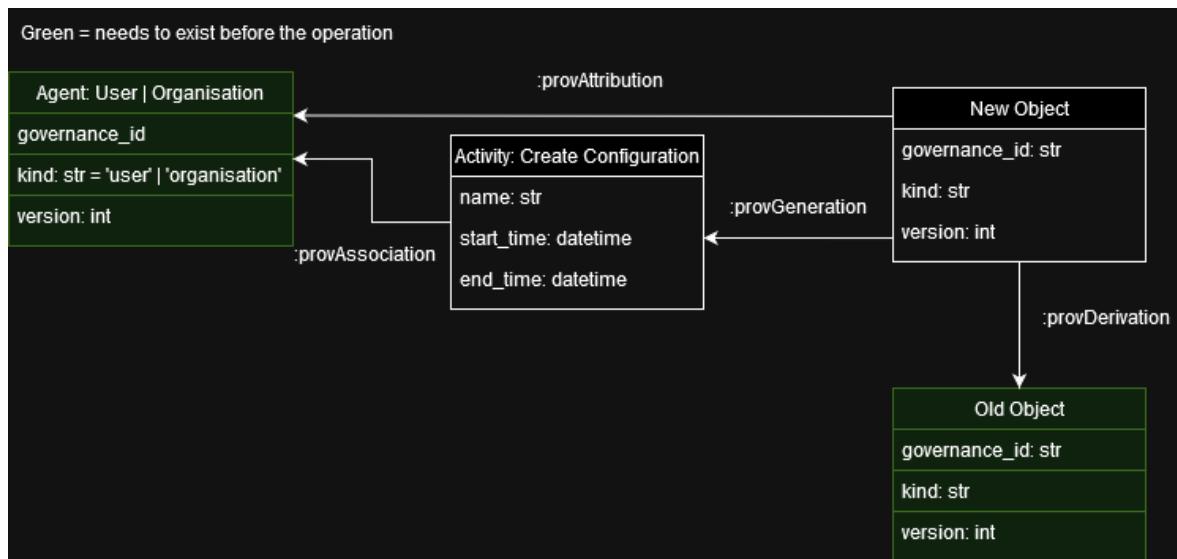


The entries in the database are structured according to [PROV-O](#) and contain the following data:

**AGENT****ENTITY****ACTIVITY**

AGENT	ENTITY	ACTIVITY
Objects	Organisation, User	Group, ML-Model, Dataset, Quality-Requirements, Proposals, Votes
Attributes	Governance-ID, Kind, Version, Timestamp	Governance-ID, Kind, Version, Timestamp
		Name, Affected_objects Start-time, End-time

Additionally, when an object is updated the following data will be created:



To accomplish this the metadata package contains three main components. The [metadata\\_middleware](#), [middleware\\_api](#) and [neo4j\\_connection](#). The middleware generates the metadata and uses [neo4j\\_connection](#) to write it into the database. The api uses [neo4j\\_connection](#) to query the database and return the result. More information can be found in the respective sections.

The package is organized in the following way:

```

ProvenanceMetadataNeo4J/
`-- metadata/
  |-- api/
  |   |-- models/
  |   |   |-- activity_model.py, agent_model.py, entity_models.py
  |   |   `-- grouped_activity_model.py, num_of_actions_model.py
  |   |-- routers/
  |   |   |-- action_router.py, config_router.py, dataset_router.py, dev_router.py
  |   |   |-- group_router.py, model_router.py, organisation_router.py
  |   |   `-- proposal_router.py, strategy_router.py, user_router.py
  |   |-- database_connection.py
  |   `-- metadata_api.py
  |-- dbmanager/
  |   |-- exceptions/
  |   |   |-- group_does_not_exist.py, object_does_not_exist.py, no_actions.py
  |   |   `-- relationship_does_not_exist.py, strategy_does_not_exist.py,
  |   |   `-- user_does_not_exist.py, version_does_not_exist.py
  |   |-- queries/
  |   |   `-- create.py, delete.py, interactions.py, retrieve.py, update.py
  |   `-- neo4j_connection.py
  |-- middleware/
  |   |-- operations/
  |   |   |-- add_config.py, add_quality_requirement.py, add_user_to_group.py
  |   |   |-- create_dataset.py, create_group.py, create_ml_model.py
  |   |   `-- create_organisation.py, create_proposal.py, create_strategy.py
  
```

```
|   |   |-- create_user.py, delete_config.py, delete_dataset.py, delete_group.py  
|   |   |-- delete_ml_model.py, delete_organisation.py, delete_proposal.py,  
|   |   |-- delete_gr.py delete_strategy.py, delete_user.py, delete_vote.py, end_voting.py  
|   |   |-- middleware_operation.py, update_config.py, update_dataset.py  
|   |   `-- update_group.py, update_model.py, update_organisation.py, update_gr.py  
|   |   `-- update_strategy.py, update_user.py vote.py  
|   |-- metadata_middleware.py  
|   `-- middleware_operations_manager.py  
|-- const.py  
`-- token.py
```

[Return to ToC](#)

## Usage

The tools provided in this package are included in the [Data Governance Full Stack](#). Therefore, installing them works the same way as installing the Full Stack.

1. Clone the [Data\\_Governance\\_Full\\_Stack](#) repository including its submodules:

```
git clone https://github.com/JsAntoPe/Data_Governance_Full_Stack.git  
cd Data_Governance_Full_Stack  
  
git submodule init  
git submodule update
```

2. Build and start the docker-compose-container:

```
docker compose build --no-cache  
docker compose up -d
```

[Return to ToC](#)

## Requirements

In order to use the tools provided in this package the following packages need to be installed:

- [fastapi](#)
- [httpx](#)
- [neo4j](#)
- [pyjwt](#)
- [uvicorn](#)

Which can be done like this:

```
pip install fastapi  
pip install httpx  
pip install neo4j  
pip install pyjwt  
pip install uvicorn
```

The documentation for these libraries can be found here:

- [fastapi docs](#)
- [httpx docs](#)
- [neo4j docs](#)
- [pyjwt docs](#)
- [uvicorn docs](#)

*Please note that docker automatically creates the virtual environment including all libraries when building the Full-Stack-Container. Therefore, when using docker it is not necessary to manually install the libraries.*

[Return to ToC](#)

## Metadata Middleware

### Description

In order to store the providence metadata there needs to be some connections between the **Data Governance Cockpit** and the database. This connection is here provided by a custom middleware for the REST-API that runs in the governance cockpit. A **middleware** is a function that gets called every time a **FastAPI** receives a request. This function is also given access to the Request- and Response-objects of the API-call. This middleware uses the information it is given to generate the provenance metadata and then store it in the database.

The middleware generates the metadata in the following way:

- Agent: governance\_id of the user that makes the API-Call
- Activity: derived from the combination of method and path of the request
- Entity: Either taken from Request- or Response-body

[Return to ToC](#)

### Requirements

The middleware uses the following packages:

- **fastapi**
- **httpx**
- **neo4j**
- **pyjwt**

Which can be installed like this:

```
pip install fastapi
pip install httpx
pip install neo4j
pip install pyjwt
```

[Return to ToC](#)

### Usage

In order to use the middleware it must be added to a FastAPI instance.

This can be done like this:

```
from fastapi import FastAPI
from ProvenanceMetadataNeo4J.metadata import MetadataMiddleware

app = FastAPI()
app.add_middleware(MetadataMiddleware)
```

[Return to ToC](#)

### Operations

The middleware supports the following operations:

Operation	Method	Path	Description
-----------	--------	------	-------------

<b>Operation</b>	<b>Method</b>	<b>Path</b>	<b>Description</b>
add_config	POST	/groups/{group_id}/strategies/{strategy_id}/configurations	adds a configurations to the given strategy
add_user_to_group	POST	/groups/{group_id}/add/{user_id}	adds a user to a group
create_dataset	POST	/datasets	creates a dataset
create_group	POST	/groups	creates a group
create_ml_model	POST	/ml-models	creates a ml-model
create_organisation	POST	/organisations	creates an organisation
create_proposal	POST	/proposals	creates a proposal
create_strategy	POST	/strategies	creates a strategy
create_user	POST	/users	creates a user
delete_config	DELETE	/groups/{group_id}/strategies/{strategy_id}/configurations/{configuration_id}	deletes a configuration
delete_dataset	DELETE	/datasets	deletes a dataset
delete_group	DELETE	/groups	deletes a group
delete_ml_model	DELETE	/ml-models	deletes a ml-model
delete_organisation	DELETE	/organisations	deletes an organisation
delete_proposal	DELETE	/proposals/{proposal_id}	deletes a proposal
delete_strategy	DELETE	/groups/{group_id}/strategies/{strategy_id}	deletes a strategy
delete_qr	DELETE	/strategies/{strategy_id}/quality_requirements/{quality_requirement_id}	deletes a quality_requirement
delete_user	DELETE	/users	deletes a users
delete_vote	DELETE	/proposals/{proposal_id}/votes/{member_id}	deletes a vote
end_voting	GET	/proposals/{strategy_id}/count_votes	ends voting
update_config	PUT	/groups/{group_id}/strategies/{strategy_id}/configurations/{config_id}/mlmodel /groups/{group_id}/strategies/{strategy_id}/configurations/{config_id}/dataset	updates a configuration
update_dataset	PUT	/datasets/{dataset_id}/features	updates a dataset
update_group	PUT	/groups/governance_id/{group_id}	updates a group
update_model	PUT	/ml-models/{model_id}	updates a model
update_organisation	PUT	/organisations/{user_id}	updates an organisation
update_qr	PUT	/strategies/{strategy_id}	updates a quality requirement
update_strategy	PUT	/strategies/{strategy_id}/quality_requirements/{quality_requirement_id}	updates a strategy
update_user	PUT	/users/{user_id}	updates a user
reset	POST	/dev/reset	deletes all database entries
vote	POST	/proposals/{proposal_id}/votes	logs a vote for a proposal

[Return to ToC](#)

## Neo4JConnection

### Description

The class `Neo4JConnection` manages access to the Neo4j-database and store the provenance metadata in the database. It contains implementations for database operations that are used by either the middleware and the API.

[Return to ToC](#)

### Requirements

`Neo4JConnection` uses the following packages:

- `httpx`
- `neo4j`

Which can be installed like this:

```
pip install httpx
pip install neo4j
```

[Return to ToC](#)

### Usage

The `Neo4JConnection` class can be used like this

1. Import `Neo4JConnection`
2. Connect to the database via the constructor
3. Use the implemented operations
4. Close the database connection

One way to use the `Neo4JConnection` class, would be like this:

```
from metadata.dbmanager.neo4j_connection import Neo4JConnection

db: Neo4JConnectionMW = Neo4JConnection('neo4j://127.0.0.1:7687')
db.reset()
db.create_user('example_user')
print(db.get_all_users())
db.close()
```

[Return to ToC](#)

### Operations

`Neo4JConnection` implements the following public basic functions:

Signature	Parameters	Return-value	Description
<code>__init__</code>	<code>uri: str</code> <code>authorization: tuple = None</code> <code>None</code> <code>database: str = None</code>		constructor for <code>Neo4JConnection</code> , creates connection to the database
<code>close</code>	<code>none</code>	<code>bool: success</code>	closes the connection to the database
<code>reconnect</code>	<code>none</code>	<code>none</code>	reconnects to the database

`Neo4JConnection` implements the following public functions to query the database:

Signature	Parameters	Return-value	Description
-----------	------------	--------------	-------------

Signature	Parameters	Return-value	Description
get_actions	user_id: str start_time: datetime = None end_time: datetime = None fetch_user_info: bool = True	list[dict]: actions	returns all actions taken by a user if one is given. if a start_time is given then it will return all actions taken since if an end_time is given it will return all actions taken until if both a start and end_time are given all actions in that interval will be returned or raises a <a>UserDoesNotExistException</a> if user_id is not an existing user
get_actions_for_object	gov_id: str	list[dict]: actions	returns all activities for the given object
get_all_datasets	none	list[dict]: datasets	returns all datasets as a list
get_all_groups	none	list[dict]: groups	returns a list of all groups
get_all_models	none	list[dict]: models	returns all models as a list
get_all_organisations	none	list[dict]: organisations	returns a list of all organisations
get_all_proposals	none	list[dict]: proposal	returns a list of all proposal
get_all_strategies	none	list[dict]: strategies	returns all strategies for a given group as a list
get_all_users	None	list[dict]: users	returns a list of all users
get_configurations	strategy_id: str	list[dict]: strategies	returns all configurations for a given strategy as a list or raises a <a>StrategyDoesNotExistException</a> if strategy_id does not exist
get_members_of_group	group_id: str	list[dict]: members	returns the members of the given group or raises a <a>GroupDoesNotExistException</a> if the group does not exist.
get_more_than_actions	num: int, start_time: datetime = None, end_time: datetime = None	list[dict]: more_than	returns a map of user and number of actions taken within the given interval for all user that have taken more than num actions
get_nodes_by_relationship	gov_id: str, relationship: str	list[dict]: nodes	returns all objects associated with the given object via the given relationship or raises a <a>ObjectDoesNotExistException</a> if the gov_id is invalid or raises a <a>RelationshipDoesNotExistException</a> if the relationship is invalid
get_num_of_actions	start_time: datetime = None, end_time: datetime = None	list[dict]: num_of_actions	returns a map of user and number of actions taken within the given interval.

<b>Signature</b>	<b>Parameters</b>	<b>Return-value</b>	<b>Description</b>
<code>get_user</code>	<code>governance_id: str</code> <code>version: int = None</code>	<code>list[dict]: user</code>	returns the current version of a user or the version that was specified or raises a <code>UserDoesNotExistException</code> if user_id is not an existing user or raises a <code>VersionDoesNotExistException</code> if version of user does not exist
<code>get_qr_for_strategy</code>	<code>strategy_id: str</code>	<code>list[dict]: qrs</code>	Returns all qrs for the strategy as a list

`Neo4JConnection` implements the following public functions to manipulate the database:

<b>Signature</b>	<b>Parameters</b>	<b>Return-value</b>	<b>Description</b>
<code>add_configuration</code>	<code>config_id: str</code> <code>strategy_id: str</code> <code>group_id: str</code> <code>model_id: str</code> <code>dataset_id: str</code> <code>user_id: str</code>	<code>bool: success</code>	creates a database entry for a configuration
<code>add_quality_requirement</code>	<code>qr_id: str</code> <code>strategy_id: str</code> <code>group_id: str</code> <code>user_id: str</code>	<code>bool: success</code>	adds a quality requirement to the given strategy
<code>add_user_to_group</code>	<code>group_id: str</code> <code>user_res: str</code> <code>user_add: str</code>	<code>bool: success</code>	adds the user with the id user_add to the group with the id user_add
<code>create_dataset</code>	<code>dataset_id: str</code> <code>user_id: str</code>	<code>bool: success</code>	creates a database entry for a dataset
<code>create_group</code>	<code>group_id: str</code> <code>user_id: str</code>	<code>bool: success</code>	creates a group
<code>create_model</code>	<code>model_id: str</code> <code>user_id: str</code>	<code>bool: success</code>	creates a database entry for a model
<code>create_organisation</code>	<code>governance_id: str</code>	<code>bool: success</code>	creates an entry for an organisation
<code>create_proposal</code>	<code>proposal_id: str</code> <code>strategy_id: str</code> <code>user_id: str</code>	<code>bool: success</code>	creates a new proposal
<code>create_strategy</code>	<code>strategy_id: str</code> <code>group_id: str</code> <code>user_id: str</code>	<code>bool: success</code>	creates a database entry for a strategy
<code>create_user</code>	<code>governance_id: str</code>	<code>bool: success</code>	creates an entry for a user
<code>delete_entry</code>	<code>gov_id: str,</code> <code>user_res: str,</code> <code>type: str</code>	<code>bool: success</code>	deletes an entry
<code>delete_vote</code>	<code>prop_id: str,</code> <code>voter_id,</code> <code>user_res: str</code>	<code>bool: success</code>	deletes a vote
<code>end_voting</code>	<code>proposal_id: str</code> <code>strategy_id: str</code> <code>user_res: str,</code> <code>type: str</code>	<code>bool: success</code>	ends voting

Signature	Parameters	Return-value	Description
update_config	type: str, config_id: str, model_id: str, dataset_id: str, user_res: str	bool: success	updates either the model or the dataset of a config
vote	proposal_id: str vote: str user_id: str	bool: success	logs a vote

**Please note that when an entry is created more than just the object itself is added to the database. The entries required by PROV-O will also be created.**

**They include, but are not limited to:**

- An activity that describes the action
- An association from the activity to user\_res
- An attribution from the created entity to user\_res

**A full explanation of the organisation of the data can be found in the [introduction](#).**

[Return to ToC](#)

## Metadata-API

### Description

The Metadata-API provides means to query the metadata database. It uses a [FastAPI](#) to provide a REST-Interface.

[Return to ToC](#)

### Requirements

The API uses the following packages:

- [fastapi](#)
- [httpx](#)
- [neo4j](#)
- [uvicorn](#)

Which can be installed like this:

```
pip install fastapi
pip install httpx
pip install neo4j
pip install uvicorn
```

[Return to ToC](#)

### Usage

Run the [run\\_metadata\\_api.py](#) file, which can be done like this. Remember to change all the URL to their local variants in [const.py](#):

```
py run_metadata_api.py
```

[Return to ToC](#)

### Operations

The Metadata-API supports the following operations:

<b>Method</b>	<b>Path</b>	<b>Description</b>
GET	/actions	returns a JSONResponse containing all actions If a start_time is given then it will return all actions taken since If an end_time is given it will return all actions taken until If both a start and end_time are given all actions in that interval will be returned :param start_time: start_time of interval :param end_time: end_time of interval
GET	/actions/grouped_by_user	Same as /actions but groups result by user
GET	/actions/num	returns a JSONResponse containing a map of user and number of actions taken within the given interval
GET	/actions/more	returns a JSONResponse containing a map of user and number of actions taken within the given interval for all user that have taken more than num actions
GET	/actions/related_to/gov_id	returns a JSONResponse containing all actions related to the object
GET	/configurations	returns a JSONResponse containing all configurations
GET	/datasets	returns a JSONResponse containing all datasets
GET	/groups	returns a JSONResponse containing all groups
GET	/groups/{group_id}/members	returns a JSONResponse containing all members of the given group
GET	/ml-models	returns a JSONResponse containing all ml-models
GET	/organisations	returns a JSONResponse containing all organisations
GET	/proposals	returns a JSONResponse containing all proposals
GET	/strategies	returns a JSONResponse containing all strategies
GET	/strategies/{id}/qr	returns a JSONResponse containing all quality requirements for the given strategy
GET	/users	returns a JSONResponse containing all users
GET	/users/{relationship}/to/{user_id}	returns a JSONResponse containing all agents connected to the object with the given user_id by the given relationship
GET	/users/{user_id}/actions? start_time&end_time	returns a JSONResponse containing all actions taken by the given user if no start_time and no end_time is specified
		returns a JSONResponse containing all actions taken by the given user since the start_time if a start_time is given
		returns a JSONResponse containing all actions taken by the given users before the end_time if an end_time is specified
		returns a JSONResponse containing all actions
GET	/users/{user_id}?version	returns a JSONResponse containing the given version of a user or the current version if version is not specified

[Return to ToC](#)