Data Tracker

SciLifeLab Data Centre

Feb 12, 2020

CONTENTS

CHAPTER

ONE

DATA STRUCTURE

The Data Tracker is based on a few main components:

- Order
- Dataset
- Project
- User
- Log
- DOI

1.1 Terminology

- Fields:
  - Fields in the documents for the datatype/collection
- Computed fields:
  - Values that are either calculated or retrieved from documents in other collection(s)
  - Included when the entity is requested via API

1.2 Order

- Added automatically when e.g. order in order portal changes to `accepted`
  - Import data from order portal
- Can have any number of associated datasets

1.2.1 Fields

- `_id`
  - Uuid for the order
- `Title`
  - Name
- `Description`
  - Description in markdown
- Creator (facility name)
  - Creator can be set to e.g. external if a non-facility wants to add a dataset
- `Receiver`
  - Email or uuid of the user who made the order
  - Input to add will be an email address, which is mapped to the user collection
  - Uuid saved: user exists
  - Email saved: user does not exist
- `Datasets`
  - All datasets generated for the order
- `Extra fields`
  - Custom fields in the style `{'key': 'key_name', 'value': 'data value'}`

1.3 Dataset

- Data generated by e.g. facility
- One per "data delivery" from facility
- Can have identifier(s) (e.g. DOIs)
- Data links can be added by `receiver`
- `Receiver` and `creator` can edit the entry (inherited from `order`)

1.3.1 Fields

- `_id`
  - Uuid of the dataset
- `Title`
  - Name
- `Description`
  - Description in markdown
- `Links`
  - List of links to where the dataset can be found.

    * List entry:

```
{
  'title': 'name',
  'url': 'https://place',
  'hashes': {
    'type': 'sha256',
    'files': [
      {
        'name': 'filename',
        'hash': 'FEDCBA9...'
      }
    ]
  }
}
```

    * `title` and `url` are mandatory for each link, `hashes` is optional

- `Extra`
  - Custom fields in the style `{'key': 'key_name', 'value': 'data value'}`

1.3.2 Computed fields

- `Related`
  - All other datasets from the same order
- `Projects`
- `Identifiers`
  - Local identifier
  - DOIs
- `Creator`
  - Inherited from `order`
  - Name of e.g. facility that generated the dataset

1.4 Project

- Created by users
- Can have multiple owners
- Can have identifiers
- Intended as a way for a user to have a page to show off their data and be able to get an identifier (DOI)

1.4.1 Fields

- `_id`
  - Uuid for the projects
- `Title`
  - Name
- `Description`
  - Description in markdown
- `Contact`
  - Contact information (email) for the project
- `Datasets`
  - Datasets connected to the dataset
  - Can be added by `receiver`/`creator` of dataset
  - Can be removed by any user listed in `owners`
- `Publications`
  - List of publications related to the project
    - Entry:

```
{
  'title': 'name',
  'doi': 'doi-id'
}
```

    - `title`+`doi` mandatory, but maybe include ability to add e.g. `journal`, `year` etc
- `DMP`
  - Data management plan
  - URL
- `Owners`
  - List of `uuids`/`emails`
  - Just like with `order`; `email` can be used if user not in db yet
    - Allow facilities to prepare project pages
- `Extra fields`
  - Custom fields in the style `{'key': 'key_name', 'value': 'data value'}`

1.4.2 Computed fields:

- `Identifiers`
  - Local identifier
  - DOIs

1.5 User

- Everyone using the system is a user
- Login via Elixir AAI
- On first login, the user will be added to db
  - Use `auth_id` to recognize user
  - Read e.g. `email` from the login info
- API can also be accessed using an API key
  - may be created by any user
- "admin" can create user for facility
- A user can "claim entries"
  - Will check all order `receivers`/project `owners` whether the users email is listed
    - `Email` will be replaced with user `uuid`
- Facilities cannot log in via Elixir, but must do so via `api_key`

1.5.1 Fields

- `_id`
  - Uuid for the order
- `Email`
  - Email address of the user
- `Auth_id`
  - Identifer received from Elixir
  - Is set to `--facility--` for facilities to avoid Elixir login
- `Api_key`
  - Key that can be used as an alternative to login for authentication
- `Name`
  - Name of the user (can be e.g. name of facility for facility accounts)
- `Affiliation`
  - University/company etc
- `Country`
  - The country of the user

- `Permissions`
  - A list of the extra permissions the user has (see *Permissions*)

1.6 Log

- Whenever an entry (`order`, `dataset`, `project`, or `user`) is changed, a log should be written
- All logs are in the same collection
- A function is required to show changes between different versions of an entry

1.6.1 Fields

- `_id`
  - `uuid` for the log
- `Action'
  - Type of action (add, edit, or delete)
- `Data_type`
  - The collection that was modified (`order`, `dataset`, `project`, or `user`)
- `Data`
  - Add/edit: complete copy of document
  - Delete: empty
- `Timestamp`
  - The time the action was performed
- `User`
  - `Uuid` of the user performing the action

1.7 DOI

- Two collections
  - `doi_req` - Requests for a DOI
  - `doi` - Accepted DOIs
- Users can request a DOI for datasets and projects
- Upon request, data is copied to `doi_req`
- A reviewer will need to check the data for the request
  - Required fields
  - File hashes
- If accepted, the data will be copied to `doi`
- Each DOI document is a complete copy of the entire data structure that was accepted for the DOI

1.7.1 Fields (request)

- `_id`
  - `Uuid` for the request
- `Data`
  - A complete copy of all relevant data
  - A project with associated datasets will include copies of the datasets in `datasets` instead of only `uuids`
- `Status`
  - `Requested`, `Accepted`, `Rejected`
- `User`
  - User that made the request
- `Updates`
  - Mini log system

```
{
  'timestamp': <current time>,
  'new_status': 'new_status'
}
```

- `Type`
  - `dataset` or `project`
- `Comments`
  - Comments from the reviewer

1.7.2 Computed Fields (request)

- `Other_requests`
  - Other requests that have been made for the same entry
  - To allow the reviewer to see e.g. earlier comments

1.7.3 Fields (doi entry)

- `_id`
  - The DOI identifier
- `timestamp`
  - When the entry was created
- `Data`
  - The complete entry that has been accepted

1.8 Other topics

1.8.1 Identifiers

- Only uuid initially
- Can request a "fancier" local identifer for `dataset`/`project`
  - Style similar to:
    * `scilifelab.facility.orderxyz.dataset1`
    * `scilifelab.projects.title1`
- All datasets and projects can request DOI
  - The required fields will be checked if empty. If not the request will be sent for evaluation by e.g. admin

1.8.2 Permissions

- "Permission classes" used to evaluate what a user may do
  - `CREATE_ORDERS`
  - `MANAGE_USERS`
  - `EDIT_ANY_DATA`
  - `READ_OWNERS`
  - `DOI_REVIEWER`
- "Default groups"
  - Template for user, giving a specific set of permissions
  - Admin - "all"
  - Facility - "create orders"+"read ownerships"

CHAPTER

TWO

API

Base URL for the API is `<url>/api`. All API description have the base implied before the first `/`.

2.1 Order

`/order/<identifier>`
: **GET** Get information about the order with uuid `identifier`.

  DELETE Delete the order with uuid `identifier`.

  PUT Update the order with uuid `identifier`.

`/order/add`
: **GET** Get an object describing the input fields for **POST**.

  POST Add a new order.

`/order/<identifier>/addDataset`
: **GET** Get an object describing the input fields for **POST**.

  POST Add a new dataset belonging to order with uuid `identifier`.

`/order/user`
: **GET** Get a list of orders created or received by current user or `username` (if provided as parameter).

2.2 Dataset

`/dataset/<identifier>`
: **GET** Get information about the dataset with uuid `identifier`.

  DELETE Delete the dataset with uuid `identifier`.

  PUT Update the dataset with uuid `identifier`.

`/dataset/all`
: **GET** Get a list of all datasets. Can be limited by parameters.

`/dataset/user`
: **GET** Get a list of datasets created or received by current user or `username` (if provided as parameter).

2.3 Project

/project/<identifier>
GET Get information about the project with uuid `identifier`.

DELETE Delete the project with uuid `identifier`.

PUT Update the project with uuid `identifier`.

/project/all
GET Get a list of all projects. Can be limited by parameters.

/project/user
GET Get a list of projects created or received by current user or `username` (if provided as parameter).

2.4 User

/user/me
GET Get information about the current user

/user/edit
GET Update information of current user

/user/edit/<uuid>
GET Update information of user with uuid `uuid`

/user/logout
GET Log out current user

/user/login
GET Log in user via elixir

/user/all
GET Get a list of all users

/user/countries
GET Get a list of countries

CHAPTER

THREE

SYSTEM FOR DEVELOPMENT

Build and activate the containers:

```
docker-compose up
```

The system can be accessed in a web browser at `localhost:5000`.

Randomized test data can be generated by `test/gen_test_db.py`. Run it using e.g.:

```
PYTHON_PATH=backend python3 test/gen_test_db.py
```

CHAPTER

FOUR

CODE REFERENCE

4.1 app.py

4.2 config.py

Settings manager for the data tracker.

Read settings from *./config.yaml*, *../config.yaml* or from the provided path.

config.**init**(*app*)

Read settings and add them to the app config.

Parameters **app** – the Flask app

config.**read_config**(*path: str = ''*)

Look for settings.yaml and parse the settings from there.

The file is expected to be found in the current, parent or provided folder.

Parameters **path** (*str*) – The yaml file to use

Returns The loaded settings

Return type dict

Raises **FileNotFoundError** – No settings file found

4.3 dataset.py

4.4 developer.py

4.5 order.py

4.6 project.py

4.7 structure.py

Required fields for the different data types.

structure.**dataset**()
Provide a basic data structure for a dataset.

Returns the data structure for datasets

Return type dict

structure.**order**()
Provide a basic data structure for an order.

Returns the data structure for orders

Return type dict

structure.**order_validator**(*data: dict*)
Validate the content of the fields of an incoming order.

Parameters **data** (*dict*) – order to check

Raises **ValueError** – bad incoming data

structure.**project**()
Provide a basic data structure for a project.

Returns the data structure for projects

Return type dict

structure.**user**()
Provide a basic data structure for a user.

Returns the data structure for users

Return type dict

4.8 user.py

4.9 utils.py

General helper functions.

utils.**check_csrf_token**()
Compare the csrf token from the request (header) with the one in the cookie.session.

utils.**check_mongo_update**(*document: dict*)
Make sure that some fields in a document are not changed during an update.

Also make sure indata is not empty.

Parameters **document** (*dict*) – received input to update a document

utils.**convert_keys_to_camel**(*chunk*)
Convert keys given in snake_case to camelCase.

The capitalization of the first letter is preserved.

Parameters **chunk** – Object to convert

Returns chunk converted to camelCase dict, otherwise chunk

Return type

-

utils.**country_list**()
Provide a list of countries.

Returns A selection of countries.

Return type list

utils.**gen_csrf_token**() → str
Genereate a csrf token.

Returns the csrf token

Return type str

utils.**get_dataset**(*identifier: str*)
Query for a dataset from the database.

Parameters **identifier** (*str*) – the uuid of the dataset

Returns the dataset

Return type dict

utils.**get_db**(*dbserver: pymongo.mongo_client.MongoClient*) → pymongo.database.Database
Get the connection to the MongoDB database.

Parameters **dbserver** – connection to the db

Returns the database connection

Return type pymongo.database.Database

utils.**get_dbserver**() → pymongo.mongo_client.MongoClient
Get the connection to the MongoDB database server.

Returns the client connection

Return type pymongo.mongo_client.MongoClient

utils.**get_project**(*identifier: str*)
Query for a project from the database.

Parameters **identifier** (*str*) – the uuid of the project

Returns the project

Return type dict

utils.**is_email**(*indata: str*)
Check whether a string seems to be an email address or not.

Parameters **indata** (*str*) – data to check

Returns is the indata an email address or not

Return type bool

utils.**is_owner**(*dataset: str = None*, *project: str = None*)
Check if the current user owns the given dataset or project.

If both a dataset and a project is provided, an exception will be raised.

Parameters

- **dataset** (*str*) – the dataset to check
- **project** (*str*) – the project to check

Returns whether the current owns the dataset/project

Return type bool

Raises **ValueError** – one of dataset or project must be set, and not both

utils.**make_log**(*data_type: str*, *action: str*, *data: dict = None*)
Log a change in the system.

Saves a complete copy of the new object.

It is assumed that all values are curated, e.g. that data only contains permitted fields.

Parameters

- **action** (*str*) – type of action (insert, update etc)
- **data_type** (*str*) – the collection name
- **data** (*dict*) – the new data for the entry

utils.**make_timestamp**()
Generate a timestamp of the current time.

Returns the current time

Return type datetime.datetime

utils.**new_uuid**() → uuid.UUID
Generate a uuid for a field in a MongoDB document.

Returns the new uuid in binary format

Return type uuid.UUID

utils.**response_json**(*json_structure: dict*)
Convert keys to camelCase and run *flask.jsonify()*.

Parameters **json_structure** (*dict*) – structure to prepare

Returns prepared response containing json structure with camelBack keys

Return type flask.Response

utils.**str_to_uuid**(*uuid_str: str*) → uuid.UUID
Convert str uuid to uuid.UUID.

Parameters **uuid_str** (*str*) – the uuid to be converted

Returns the uuid

Return type uuid.UUID

CHAPTER

FIVE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

INDEX