

Exploring the API of 4TU.ResearchData for RDM support staff workshop (TU/e, WUR, UT)



Workshop Overview

The idea of this workshop is to familiarize them with the main endpoints of the WebAPI for requesting, downloading, searching, and uploading datasets and software to the 4TU.ResearchData repository. The main target audience is the RDM support staff from the Dutch technical universities

The proposed schedule for a 3-hour session is:

- Introduction (10 minutes)
- Hands-on practice with the endpoints in the terminal (up to 2 hours, including one break)
- Discussion and troubleshooting: time for questions and for addressing projects, issues, or challenges participants may already have within their faculties (1 hour)



Schedule

Main topics:

- Fetching datasets, software, collections
- Searching and fetching data from authors
- Searching accounts within your institution
- Uploading
 - Metadata uploading
 - File uploading
 - Submit for review
- Image preview

Prerequisites

Before attending the workshop, please ensure you have:

- (Required) Two private tokens in data.4tu.nl (main environment) and in next.data.4tu.nl (test environment)
- (Required) The `yq` a command-line YAML processor (similar syntax to `jq`).

- Linux

```
sudo apt-get update
sudo apt install yq
```

- macOS

```
brew install yq
```

- Windows (in powershell)

- Install scoop

```
1 | Set-ExecutionPolicy -ExecutionPolicy RemoteSigned -Scope CurrentUser
2 | Invoke-RestMethod -Uri https://get.scoop.sh | Invoke-Expression
```

```
scoop install yq
```

or

```
choco install yq
```

- (Optional but nice to have) The `jq` program to render nicely json outputs:

- Linux

```
sudo apt-get update
sudo apt-get install -y jq
```

- macOS

```
brew install jq
```

- Windows Powershell

- Install scoop

```
1 | Set-ExecutionPolicy -ExecutionPolicy RemoteSigned -Scope CurrentUser
2 | Invoke-RestMethod -Uri https://get.scoop.sh | Invoke-Expression
```

```
1 | scoop install main/jq
```

- Check in Git bash your installation

```
1 | yq --version
2 | jq --version
```

- If you're using Git Bash or WSL, use the Linux instructions above.
- Otherwise, download the executable from the jq releases page (<https://jqlang.org/download/>) and add it to your PATH.

Introductory presentation

Presentation link : <https://zenodo.org/records/17520532>

4TU API v2 documentation

- Incomplete!!!
- <https://djhuty.4tu.nl/#x1-630006>
- We have developed the v3 with new features related to:
 - Git repositories
 - Data upload
 - Image preview and modification via IIIF
 - Grups id for specific institutions
 - RO-CRATE (Research object crate)

Figshare documentation

v2 of the API of 4TU is backward compatible with the API from Figshare

https://docs.figshare.com/#articles_list

Thus the same endpoints , parameters and response schema apply to both.

Why is it important to use Web APIs for research?

Web APIs help automate access to research data and metadata. This enables reproducibility, automation of data pipelines, and programmatic interaction with repositories like 4TU.ResearchData.

REST APIs in a nutshell

A REST API is a web service that uses HTTP methods (GET, POST, etc.) to allow communication between clients and servers. Responses are usually in JSON format, making them easy to parse and reuse.

List of commands

Reuse: Search and Download Datasets

Get datasets or software deposited in 4TU (via `curl`)

```
curl -X GET "https://data.4tu.nl/v2/articles" | jq
```

What is curl?

curl stands for **C**lient **U**RL.

It's a command-line tool that allows you to transfer data to or from a server using various internet protocols, most commonly HTTP and HTTPS.

It is especially useful for making API requests — you can send GET, POST, PUT, DELETE requests, upload or download files, send headers or authentication tokens, and more.

Why curl works for APIs

REST APIs are based on the HTTP protocol, just like websites. When you visit a webpage, your browser sends a GET request and displays the HTML it gets back. When you use curl, you do the same thing, but in your terminal. For example:

```
curl https://data.4tu.nl/v2/articles
```

 This sends an HTTP GET request to the 4TU.ResearchData API.

Key reasons why curl is used:

It's built into most Linux/macOS systems and easily installable on Windows.

Scriptable: usable in bash scripts, notebooks, automation.

Supports headers, query parameters, tokens, POST data, etc.

Can output to files (>, -o, -O) or pipe to processors like jq.

Add parameters to the same endpoint to filter results

- open the documentation: <https://djarahuty.4tu.nl/> (apologies by the documentation, it is also incomplete , we will try to make it better...)

```
curl "https://data.4tu.nl/v2/articles?limit=2&published_since=2025-05-01" > data.json
```

```
curl "https://data.4tu.nl/v2/articles?limit=2&published_since=2025-05-01" | jq
```

Exercise : request 10 datasets published from January 1st 2025 and show it in the screen

```
curl "https://data.4tu.nl/v2/articles?item_type=3&limit=10&published_since=2025-01-01" | jq
```

Tip: The v2 of the API of 4TU.ResearchData is based on the figshare API , which practically means, that if you dont find something you were looking for in the current documentation <https://djarahuty.4tu.nl/#x1-640006.1>, you can look in : https://docs.figshare.com/#articles_list

Get 10 software records published after 01-01-2025 (via curl)

```
curl "https://data.4tu.nl/v2/articles?item_type=9&limit=1&published_since=2025-01-01" | jq
```

Get information per dataset ID

```
curl "https://data.4tu.nl/v2/articles/03c249d6-674c-47cf-918f-1ef9bdafef749" | jq #
```

Get all the files per dataset ID

```
curl "https://data.4tu.nl/v2/articles/03c249d6-674c-47cf-918f-1ef9bdafef749/files" | jq

## Open this link in the browser to check the uuid of a file to download (the readme file)
```

How to download a specific file

```
# print the readme file in the screen
```

```
curl "https://data.4tu.nl/file/03c249d6-674c-47cf-918f-1ef9bdafef749/20382d28-0ed9-4f7b-b1d1-1a1a1a1a1a1a" -O
```

Command	Behavior
<code>curl URL`</code>	Prints file to screen (no saving)
<code>curl -O URL`</code>	Downloads and saves with original name
<code>curl -o filename URL`</code>	Downloads and saves with custom name
<code>curl -L -O URL`</code>	Follows redirects and saves file

```
#| `curl -C - -O URL` | Resumes an interrupted download |
```

Collections

Fetching all collections

```
curl "https://data.4tu.nl/v2/collections" | jq
```

Fetching collections with parameters

```
curl "https://data.4tu.nl/v2/collections?limit=2&published_since=2025-01-01" | jq
```

Fetching information of a specific collection

```
curl "https://data.4tu.nl/v2/collections/a72aa7ae-7fd2-450b-a1c4-1fa093d15438" | jq
```

Fetching information of the datasets of a collection

```
curl "https://data.4tu.nl/v2/collections/a72aa7ae-7fd2-450b-a1c4-1fa093d15438/articles"
```

Search Datasets by Keyword

```
curl --request POST --header "Content-Type: application/json" --data '{ "search_for"
```

```
curl --request POST --header "Content-Type: application/json" --data '{ "search_for"
```

Using a Token to Access Author Info (via `curl`)

Create the `.env` file and copy your private token there

```
echo 'API_TOKEN="your_token_here"' > .env
```

```
echo "Token loaded: ${API_TOKEN:0:5}..."
```

```
source .env
```

Searching authors by name

```
# Requires setting a token in a sourced .env file (maybe skip this step but mention
curl --request POST https://data.4tu.nl/v2/account/authors/search --header "Authoriz
```

Searching accounts within your institution

- "<https://data.4tu.nl/v3/groups>" (GET) This endpoint lists the groups ID of the institutions that you can use to filter the output of v2/articles/

```
curl -X GET "https://data.4tu.nl/v3/groups" | jq
```

Upload Datasets (POST Requests)

Basic Upload of metadata to a draft dataset

```
curl -X POST https://next.data.4tu.nl/v2/account/articles --header "Authorization:
```

Adding an author to the draft dataset

- first we need to copy the uuid of the draft dataset created in the previous step in the next.data.4tu.nl website

```
curl -X POST "https://next.data.4tu.nl/v2/account/articles/UUID/authors" --header "A
```

Upload Using YAML Metadata

- They need to download the example_metadata.yaml file

```
curl -o example_metadata.yaml https://raw.githubusercontent.com/4TUResearchData-Carpentries/WebAPI4RDM/refs/heads/main/Lesson_development/example_metadata.yaml
```

Upload to next server

```
yq '.' example_metadata.yaml | curl -X POST https://next.data.4tu.nl/v2/account/arti
```

Upload to the production server

```
yq '.' example_metadata.yaml | curl -X POST https://data.4tu.nl/v2/account/articles
```

Command explanation:

`yq '.' example_metadata.yaml` : Converts example_metadata.yaml into JSON

- yq is a command-line tool to read/manipulate YAML (like jq is for JSON).

- `'.'` means "read the full YAML structure as-is".

```
-d @-
```

- `-d` sends data in the body of the POST request.
- `@-` means: read the request body from stdin (standard input), i.e., the piped-in JSON from yq.

Now try to submit it and realize that need a least a file to submit for review

File upload

```
curl -X POST "https://next.data.4tu.nl/v3/datasets/dataset-id/upload" --header "Au
```

Now lets take the uuid of the draft just created in the previous example and put it in the endpoint

- For the data, first download the data using curl from github

```
curl -O "https://raw.githubusercontent.com/4TUResearchData-
```

```
Carpentries/WebAPI4RDM/refs/heads/main/Lesson_development/data_files/test_a.csv"
```

```
curl -X POST "https://next.data.4tu.nl/v3/datasets/UUID/upload" --header "Authoriz
```

File upload with strict check for empty files and duplicates

```
MD5SUM=$(md5sum "ABSOLUTE_PATH2FILE" | awk '{print $1}')
```

```
curl -X POST "https://next.data.4tu.nl/v3/datasets/UUID/upload?strict_check=1&md5=${
```

the response of this is that the resource is already available and stops there

Submit for review

```
yq '.' example_metadata.yaml | curl -X PUT "https://next.data.4tu.nl/v3/datasets/UUID
```

Image preview with IIIF (International Image Interoperability Framework)

IIIF is a standard for delivering high-resolution images over the web. It allows users to zoom, pan, and interact with images in a web browser.

Open the browser and insert this endpoint:

<https://data.4tu.nl/iiif/v3/c3eee5e4-1651-4541-8fb4-f240fbd1c4ba/full/1024,1024/0/default.jpg>

Explore the manifest

- example: <https://data.4tu.nl/iiif/v3/c3eee5e4-1651-4541-8fb4-f240fbd1c4ba/full/1024,1024/0/default.jpg>
 - dataset of the example : <https://data.4tu.nl/datasets/8289a903-7ccf-401b-af66-f5b3c9abe4b6/1>
- "https://data.4tu.nl/iiif/v3/<file_uuid>" (GET, context of the image)
 - example: <https://data.4tu.nl/iiif/v3/312f1d4a-2b83-491c-b906-a9d5497f6c9d>
- "<https://data.4tu.nl/iiif/v3/c2a8d5ce-c4ea-46ed-bcdc-e35033e908a8/1/manifest>"
 - To open it in an editor (<https://manifest-editor.digirati.services/?tab=recent>)

Motivation for Using bash :

Use case: Imagine a researcher is interested in getting the descriptions and categories of datasets uploaded in April 2025

Challenge: The description and categories are exposed if a dataset in specific is queried

```
curl -s "https://data.4tu.nl/v2/articles/fb26fd3f-ba3c-4cf0-8926-14768a256933" | jq
```

Get the description and categories of the datasets uploaded in April 2025

```
curl -s "https://data.4tu.nl/v2/articles/fb26fd3f-ba3c-4cf0-8926-14768a256933" | jq
```

Bash Script: Loop Through UUIDs to Collect Metadata

```
curl -s "https://data.4tu.nl/v2/articles?published_since=20250401&item_type=3&limit="
```

Limitations of Bash Scripts

- Harder to debug or extend
- Tricky to structure or merge data
- Not ideal for large-scale automation

Bonus: Using `connect4tu` bash Package

You can also use the [connect4tu](#) package for a cleaner bash interface to the 4TU API.