ARC-Finder – A simple, locally-deployed tool to find your peer's research data

Projektmodul (Modul 9) des Zertifikatskurs FDM (15.07) 2021 / 2022

Dominik Brilhaus,
https://orcid.org/0000-0001-9021-3197

2022-06-09

Contents

Introduction

Motivation

Research is a highly collaborative endeavor that builds on the synergistic interaction between different stakeholders enabled by efficient knowledge exchange. Gaining a prompt overview of the ongoing research efforts – both pre- and post-publication – is oftentimes hindered for social, legal or technical reasons. This often holds true even between parties of spatially closest and well trusted surroundings of a collaborative consortium such as the Cluster of Excellence on Plant Sciences (CEPLAS[1]). The key to enable discussion on and exchange of research data is *findability*, the first layer of the FAIR principles[2] of data stewardship (Wilkinson *et al.*, 2016[3]). The project presented here aims to address this layer, by making CEPLAS research easily findable and visible amongst CEPLAS researchers and showcase the beauty and ease of data sharing to spike fruitful collaborations with peers.

DataPLANT and the Annotated Research Context

Research data management (RDM) within CEPLAS is closely aligned with DataPLANT[4], the NFDI[5] consortium for plant sciences. At the heart of DataPLANT's RDM strategy lies the Annotated Research Context (ARC[6]), a directory structure that packages research data together with associated metadata and computational workflows into self-sustained research objects. Annotation of research data in the ARC is based on the metadata schema ISA[7] (for investigation – study – assay). Serialized in spread sheet format as *ISA-tab* this enables intuitive, flexible and yet structured and conclusive metadata annotation of the versatile data types produced in plant sciences. ARCs are git[8] repositories that can be shared via DataPLANT's DataHUB[9], a customized GitLab[10] instance with a federated authentication interface to allow controlled access across institute borders.
Although the ARC environment is continuously being developed, the choice of these key technical pillars are set: (a) ARC as the structure, (b) ISA as the metadata language, (c) git as version control logic and (d) gitlab for ARC collaboration and user management. This allows to leverage the ARC environment and develop (intermediate) solutions for data findability, knowing that time and efforts are well-invested, since both (meta)data ingest into as well as secondary outputs dependent on the

[1] CEPLAS, https://ceplas.eu
[2] GO-FAIR, https://www.go-fair.org/fair-principles/
[3] Wilkinson, M., Dumontier, M., Aalbersberg, I. et al. The FAIR Guiding Principles for scientific data management and stewardship. Sci Data 3, 160018 (2016). https://doi.org/10.1038/sdata.2016.18
[4] DataPLANT, https://nfdi4plants.de
[5] Nationale Forschungsdaten Infrastruktur, https://www.nfdi.de/
[6] ARC specifications, https://github.com/nfdi4plants/ARC-specification/
[7] ISA Metadata Schema, https://isa-tools.org/
[8] Git, https://git-scm.com/
[9] DataPLANT DataHUB, https://git.nfdi4plants.org
[10] GitLab, https://gitlab.com

ARC will be adoptable and migratable in the future.
While (contents of the) ARCs can be searched via standard GitLab-implemented mechanisms within the DataHUB or via standard routines on a user's system where the ARCs are locally cloned and stored, a structured and user-friendly search interface tailored to metadata stored in multiple ARCs – including unpublihed ARCs – is currently unavailable. With the ARC-Finder presented here, I seek to close this gap with a lightweight quickfix.

Implementation

Technical back-end

The technical back-end of the ARC-Finder is a combination of shell and R scripts. For data retrieval it leverages the GitLab API [11]. The GUI is based on RStudio's ShinyApp[12]. The design idea was to rely on as few programming language environments as possible. The actual code work is attached in the supplemental materials (see scripts) and available online (see availability). Software dependencies are listed in the supplemental materials (see dependencies).

The ARC-Finder workflow

The ARC-Finder employs three concerted, but independent modules of metadata retrieval, restructure, and representation (Fig. 1).

The ARC-Finder can be run in two modes. If the user does not supply a gitlab personal access token (PAT), the ARC-Finder retrieves metadata only from publicly accessible ARCs. If a functional PAT is provided by the registered user, metadata is retrieved from both public and privately shared ARCs. For detailed user instructions see README.md. The ARC-Finder selectively scans all user-accessible ARCs only for the ISA investigation workbooks (`isa.investigation.xlsx`) stored at the root of every ARC. The identified workbooks are downloaded and dumped locally in a temporary folder on the user's machine. Next, the ARC-Finder restructures the investigation-level metadata into a simple spreadsheet-based database. From the database the metadata is fed into and represented by the ARC-Finder graphical user interface (GUI).

[11] GitLab Application Programming Interface (API), https://docs.gitlab.com/ee/api/
[12] ShinyApps, https://www.shinyapps.io/

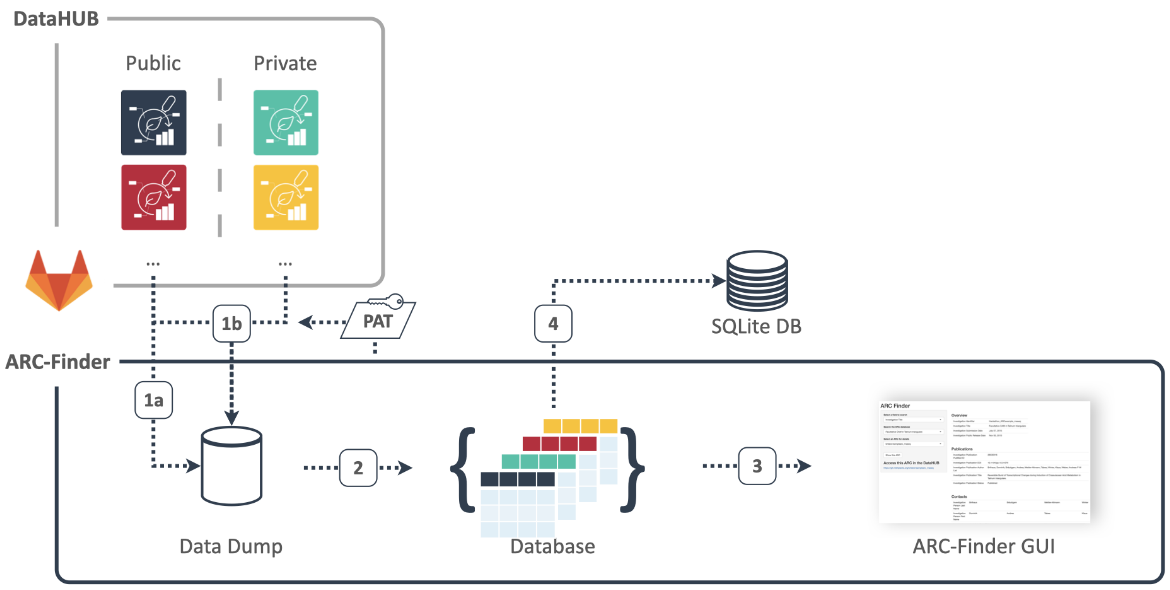
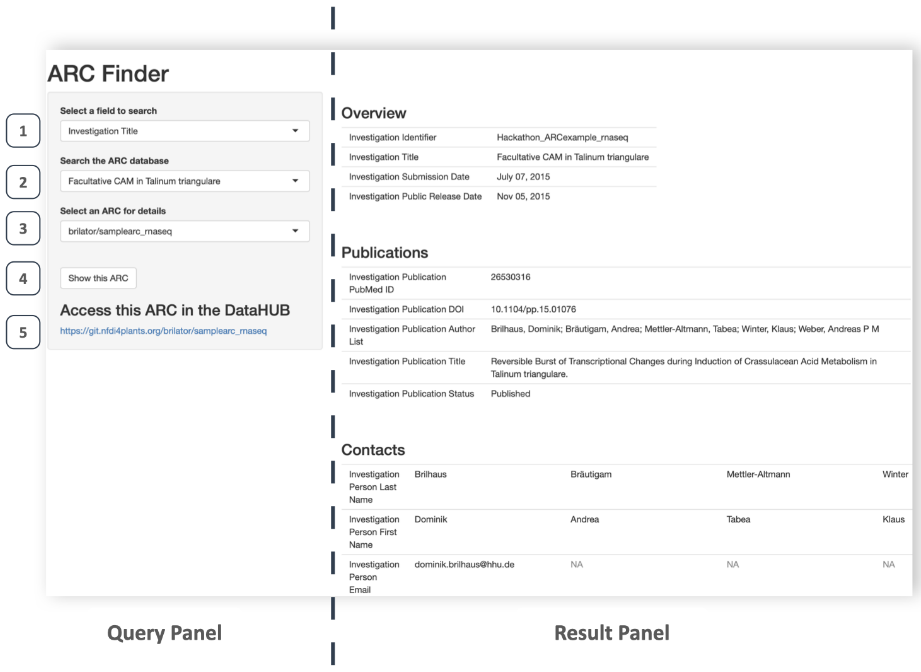


Figure 1: The ARC-Finder Workflow. Depending wether the user provided a gitlab personal access token (PAT). the ARC-Finder retrieves publicly (1a) or publicly and privately (1b) accessible metadata from the DataHUB and stores it in a local data dump. The metadata is restructured into a searchable database (2) and fed into the ARC-Finder graphical user interface (GUI) for clear representation (3, details see Fig. 2) as well as provided as an SQLite database (DB) (4).

The ARC-Finder GUI

The ARC-Finder GUI is a responsive ShinyApp running in the user's default web browser (Fig. 2). Three dropdown search fields build the core of the GUI's *Query Panel*. The user can select to search any or a specific metadata attribute (Fig. 2 - Field 1) for specific terms provided as free-text or selected from the search field (Fig. 2 - Field 1). Matching ARCs are listed for selection in a dropdown menu (Fig. 2 - Field 3). Once an ARC is selected, a click on the "Show this ARC" button (Fig. 2 - Field 4) reveals the metadata associated with the ARC in the *Result Panel* and provides a link to the respective ARC in the DataHUB (Fig. 2 - Field 5).



Select a field to search: Investigation Title

Search the ARC database: Facultative CAM in Talinum triangulare

Select an ARC for details: brilator/samplearc_rnaseq

Access this ARC in the DataHUB

https://git.nfdi4plants.org/brilator/samplearc_rnaseq

Overview

Investigation Identifier	Hackathon_ARCexample_rnaseq
Investigation Title	Facultative CAM in Talinum triangulare
Investigation Submission Date	July 07, 2015
Investigation Public Release Date	Nov 05, 2015

Publications

Investigation Publication PubMed ID	26530316
Investigation Publication DOI	10.1104/pp.15.01076
Investigation Publication Author List	Brilhaus, Dominik; Bräutigam, Andrea; Mettler-Altmann, Tabea; Winter, Klaus; Weber, Andreas P M
Investigation Publication Title	Reversible Burst of Transcriptional Changes during Induction of Crassulacean Acid Metabolism in Talinum triangulare.
Investigation Publication Status	Published

Contacts

Investigation Person Last Name	Brilhaus	Bräutigam	Mettler-Altmann	Winter
Investigation Person First Name	Dominik	Andrea	Tabea	Klaus
Investigation Person Email	dominik.brilhaus@hhu.de	NA	NA	NA

Figure 2: The ARC-Finder GUI is divided into two panels. The user can search for available ARCs in the *Query Panel* (left, details elaborated in the text). Investigation-level metadata of the selected ARC is presented in the *result panel* (right).

Discussion

With the ARC-Finder presented here, I tried to tackle a common challenge of RDM within CEPLAS (and likewise many other collaborative research consortia): easy and structured findability of research data of peers, including unpublished datasets that are just in the making. The ARC-Finder is built on the developments within DataPLANT surrounding the ARC environment and especially relies on the ISA metadata model and the GitLab-backed DataHUB.

Leveraging on the ARC environment, the ARC-Finder follows a comparably straight-forward approach and yields an instantaneous benefit to the researcher. Metadata provided by the user to collaboration partners via the DataHUB becomes immediately searchable via the ARC-Finder. While advocating FAIR data stewardship to the users (e.g. plant researchers), one of the major hurdles is the continuously changing plethora of platforms and tools offering one or the other RDM service (supposedly in a better way than competitors). This can range from a variation of electronic lab notebooks, cloud services, wikis, repositories or even chat software, leaving the researcher frustrated and unwilling to use (or adopt to) yet another RDM platform in the future. The ARC-Finder show-cases how the use of established standards, the ISA metadata model and git, facilitates extensibility and boosts sustainable RDM. Even if the tool itself may not see a long-term interest, it serves a quick benefit, while all metadata provided to the DataHUB will be integrable with future developments, including a more sophisticated metadata registry. This avoids user friction and makes the ARC environment more appealing to the researchers. To provide an example for an alternative output, the ARC-Finder stores the structured data as an SQLite database (termed `yourARCs_database.sqlite`) for use in third party applications, Fig. 1 - Field 4).

While the ARC-Finder can list unpublished and thus possibly sensitive data, it does not itself handle any user rights. Data safety and access management depends on the authentication mechanisms provided by the DataHUB. Here, access to the ARCs can be controlled to share them publicly or with invited collaborators. Still, by design the ARC-Finder focuses on metadata at the highest project and least sensitive (i.e. ISA's "investigation") level to minimize possible discomfort with data sharing.

The simple design of the ARC-finder comes with a few caveats and leaves room for future improvements. For reasons of simplicity and data safety (see above), the depth of metadata findability is limited to the investigation-level, ignoring the biologically more relevant study and assay levels of the ISA model. Furthermore, only those ARCs shared by individual users (not groups) are included and the ARC-Finder only searches the default `main` branch of the ARCs git backend. Both limitations could be easily extended in future versions of the ARC-Finder.

Several design decisions limit the ARC-finder's efficiency and scalability. The ARC-Finder is solely deployed locally and does not store any data or interaction on a server-side. Every time the ARC-Finder is run, it removes and overwrites the temporary database from earlier runs rather than updating or appending to it. In the current version, the user cannot make any pre-selection about the scope of ARCs to be searched, e.g. only ARCs associated to a specific user or group. Thus every ARC-Finder run scans and retrieves data from all available ARCs. Two paralleling serializations of ISA metadata exist in the ARC. The user-centered ISA workbooks (e.g. `isa.investigation.xlsx`) allow for intuitive metadata annotation. However, depending on the complexity of the ARC as well as the user input, these files can easily become relatively big adding to the efficiency issue. For programmatic interaction all ISA metadata in the ARC can be exported to the lightweight and less error-prone JSON format (termed `arc.json`). The decision to center the ARC-Finder around the `isa.investigation.xlsx` workbook was made to spare another dependency detour to convert between formats.

Supplemental Material

Availability

The ARC-Finder is available for download at https://github.com/Brilator/arcFinder.

Dependencies

Software

Table 1: Software used during development, testing and writing.

Software	Version	Platform
GNU bash	3.2.57(1)-release	x86_64-apple-darwin21
curl	7.79.1	x86_64-apple-darwin21.0
R	4.2.0	x86_64-apple-darwin17.0
RStudio	2022.02.2 Build 485	-
Visual Studio Code	1.67.2	-
Codes Spell Checker (VS Code Extension)	2.03	-
pandoc	2.18	-
TeX Live 2022	MacTeX-2022	-

R libraries

To provide best reproducibility, R package dependencies are handled via `renv`[13] (version 0.15.3) and stored in the root file "renv.lock". In the first step of `arcFinder`, the virtual environment is automatically restored, including installation of all required dependencies. Depending on the local setup (installation of R and packages), this may take some time. However, `renv` prevents interference with the local setup, thus keeping the system intact.

Table 2: R packages specifically loaded for individual R scripts

Package (version)	Main purpose	Used in script(s)
renv_0.15.4	Manage R package dependencies	01_install_dependencies.R 01_restore_dependencies.R

[13] R package "renv", https://rstudio.github.io/renv/

Table 2: R packages specifically loaded for individual R scripts

Package (version)	Main purpose	Used in script(s)
readxl_1.4.0 (part of 'tidyverse')	Read data from Microsoft Excel workbooks	03_parse_isaInvxlsx.R
tidyverse_1.3.1	Tidy data into a useful format	04_searchApp/app.R
shiny_1.7.1	Prepare and launch a shiny app	04_searchApp/app.R
DBI_1.1.2	Write data to an '*.sqlite' object	05_pull_together_sql.R

Platform

The DataPLANT's DataHUB[14] is a customized instance of GitLab[15], currently running under version 14.10.2, hosted and maintained by the DataPLANT node at Albert-Ludwigs-University Freiburg. Data is retrieved from the DataHUB via GitLab API version 4.
After registration[16] with DataPLANT, users can share and access non-public ARCs via the DataHUB. As explained in the `arcFinder`'s README, a GitLab private access token (PAT) needs to be generated within the DataHUB and provided to `arcFinder`.

Tests

The ARC-Finder was currently tested only under macOS Monterey 12.3.1 (x86_64-apple-darwin17.0, 64-bit) with software versions specified under Dependencies.

Deviation from the original concept

The originally proposed concept targeted an automated workflow for easier metadata-ingestion from previously published manuscripts into an the ISA model of an ARC. As this workflow (a) targets a completely other "side" of the ARC and DataHUB environment and thus (b) comes with multiple additional and more complicated dependencies, it was omitted from the ARC-Finder presented here.

[14] DataPLANT DataHUB, https://git.nfdi4plants.org
[15] GitLab, https://gitlab.com
[16] DataPLANT registration, https://register.nfdi4plants.org/

Scripts

arcFinder.sh

```
#########################################################
### Create root folder for temporary data
#########################################################

### If exists, remove and create fresh.
### This is to prevent data piling.
### TODO Should probably be replaced with safer / better logic for
    debugging. TODO

if [ -d ".tmp/" ]; then
  rm -r .tmp/
  mkdir .tmp/
else
  mkdir .tmp/
fi

#########################################################
### Resotre renv session
#########################################################

echo "### Restore virtual environment"
echo "----------------------"

Rscript ./scripts/01_restore_dependencies.R 2>&1 >> .tmp/01.log

#########################################################
### Read GitLab personal access token (PAT)
#########################################################

### Read GitLab PAT from -p flag

while getopts p: flag
do
    case "${flag}" in
        p) gitlab_pat=${OPTARG};;
    esac
done

### Check if argument supplied with `-p` is a file.
### If yes, read that file.
### If not, use the input (PAT as a string) directly

if [ -f "$gitlab_pat" ]; then
    echo "Using GitLab token stored in '$gitlab_pat'."
    gitlab_pat=$(< $gitlab_pat)
fi
```

```bash

### check if string is empty

[ -z "$gitlab_pat" ] && printf "No GitLab token supplied or GitLab
    token is empty. \nReading from public ARCs only.\n"

########################################################
### Run gitlab reader
########################################################

echo "-----------------------"
echo "### Step 01: Downloading metadata of available ARCs from the
    DataHUB."
echo "-----------------------"

echo "log of 02_read_from_gitlab.sh" > .tmp/02.log
bash ./scripts/02_read_from_gitlab.sh -p "${gitlab_pat}" 2>&1 >> .tmp
    /02.log

########################################################
### Run xlsx parser
########################################################

## store paths of isa.investigation.xlsx files into variable
## while loop
## - extract arc id from part of path
## - run script with arc id and path

echo "### Step 02: Structuring ARC metadata."
echo "-----------------------"
echo "log of 03_parse_isaInvxlsx.R" > .tmp/03.log

invs=$(find .tmp/02_investigations -name '*.xlsx' | sort -n)
echo "$invs" | while IFS= read -r current_inv_path;
do
  arc_id=$(echo $current_inv_path | cut -d/ -f3 | cut -d"_" -f1)

  Rscript ./scripts/03_parse_isaInvxlsx.R "$arc_id" $current_inv_path
      2>&1 >> .tmp/03.log

done


########################################################
### Pull together data
########################################################

echo "### Step 03: Building a searchable database of ARC metadata"
echo "-----------------------"

Rscript ./scripts/03_pull_together.R 2>&1 >> .tmp/03.log
```

```

##########################################################
### Optional: Prepare SQLite database
##########################################################

Rscript ./scripts/05_pull_together_sql.R 2>&1 >> .tmp/05.log

##########################################################
### Run the search APP
##########################################################

echo "### Voila: The ARC Finder is running in your default browser."
echo "### Close the browser window or tab to shut down the app."

Rscript -e 'load(".tmp/03_allARCs.RData"); shiny::runApp("./scripts/04
    _searchApp/app.R", launch.browser = TRUE)' 2>&1 >> .tmp/04.log
```

README.md

ARC-Finder – A simple, locally-deployed tool to find your peer's research data

This is a tool to help you find metadata about ARCs stored in the DataPLANT DataHUB. Visit the DataPLANT website for more information about ARCs (annotated research contexts).

Usage

- Git clone or download this repository.
- Open a command line or terminal and navigate to the `arcFinder` directory.
- Run one of the following two options:

Option 1: Search public ARCs only

```
./arcFinder.sh
```

Option 2: Search Public + privately shared ARCs

> Note: Replace `<gitlab pat>` with the path pointing to a file which stores a GitLab personal access token (PAT).

```
./arcFinder.sh -p <gitlab pat>
```

Registration with DataPLANT In order to use the `<gitlab pat>` option, please follow these steps:

1. Sign up with DataPLANT.
2. Generate a personal access token in the DataHUB PAT settings

    - Provide a "Token name", e.g. `arcFinder`
    - Select either option "api" or "read_api" and click "Create personal access token"
    - Copy the generated token on top of the page.

3. Paste the bare token into a text file and save it (e.g. `gitlab_token` stored in the root of this directory)
4. Supply the file path to `arcFinder`, e.g.:

```
./arcFinder.sh -p gitlab_token
```

Documentation

- Read the full project outline for more details.
- Check out the gif to see the ARC-Finder in action

scripts/01_install_dependencies.R

```r
############################################################
### Script to install all R dependencies
############################################################


if(!require(tidyverse, quietly = TRUE)){install.packages("tidyverse")}
if(!require(DBI, quietly = TRUE)){install.packages("DBI")}
if(!require(shiny, quietly = TRUE)){install.packages("shiny")}
if(!require(renv, quietly = TRUE)){install.packages("renv")}

renv::init(bare = T)

# save library state to lockfile
renv::snapshot()

renv::status()
```

scripts/01_restore_dependencies.R

```r

############################################################
### Restore R virtual environment via renv
############################################################

# restore lockfile, thereby installing dependencies from renv.lock

renv::restore()
```

scripts/02_read_from_gitlab.sh

```bash
#!/usr/bin/env bash

############################################################
### Script to skim GitLab for accessible ARCs and
### retreive their isa.investigation.xlsx's
############################################################

### Goal
# 1. read gitlab pat
# 2. Store a list of all available projects (e.g. tab)
# 3. Iterate over project trees
#   - check for isa.investigation file
#   - present: download raw and dump to temp
#   - absent: leave loop
```

```bash

############################
### Read GitLab token (PAT)
############################

### Read GitLab PAT from -p flag

while getopts p: flag
do
    case "${flag}" in
        p) gitlab_pat=${OPTARG};;
    esac
done

# ### Check if argument supplied with `-p` is a file.
# ### If yes, read that file.
# ### If not, use the input (PAT as a string) directly

# if [ -f "$gitlab_pat" ]; then
#     echo "Using GitLab token stored in '$gitlab_pat'."
#     gitlab_pat=$(< $gitlab_pat)
# else
#     echo "Using supplied GitLab token"
#     # This would be gitlab_pat=$gitlab_pat   ### TODO: probably safer to change this
# fi

# ### check if string is empty

# [ -z "$gitlab_pat" ] && printf "No GitLab token supplied or GitLab token is empty. \nReading from public ARCs only.\n"


############################
### List available ARCs
############################

# Writing to json first
curl --silent --request GET --header "PRIVATE-TOKEN: $gitlab_pat" "https://git.nfdi4plants.org/api/v4/projects/" > .tmp/02_arcs_available.json

# grepping project IDs
grep -oE '"id":[0-9]{1,},"description"' .tmp/02_arcs_available.json | grep -oE '[0-9]{1,}' > .tmp/02_arcs_ids

# Could be piped directly (without the temporary .json)
# But will keep the json, for trouble-shooting
# curl --request GET --header "PRIVATE-TOKEN: $gitlab_pat" "https://git.nfdi4plants.org/api/v4/projects/" | grep -oE '"id":[0-9]{1,},"description"' | grep -oE '[0-9]{1,}' > projects_list
```

```bash


############################
### Iterate over ARCS
############################

### create a dump directory for the isa.investigation.xlsx files
if ! [ -d ".tmp/02_investigations" ]; then mkdir ".tmp/02_investigations"; fi

### write a table to collect ARC id and path with namespace
printf "ARC id\tARC path\tcomment" > .tmp/02_investigations/arc_list.tsv

all_arc_IDs=$(< .tmp/02_arcs_ids)
echo "$all_arc_IDs" | while IFS= read -r arc_id;
do
  # echo $arc_id

  ### get project info
  curl --silent --header "PRIVATE-TOKEN: $gitlab_pat" "https://git.nfdi4plants.org/api/v4/projects/$arc_id" > .tmp/02_current_arc_info.json

  ### extract git path with namespace
  arc_path=$(grep -oE '"path_with_namespace":".*,"created_at' .tmp/02_current_arc_info.json | cut -d'"' -f 4)

  echo $arc_path

  ### get project tree
  curl --silent --header "PRIVATE-TOKEN: $gitlab_pat" "https://git.nfdi4plants.org/api/v4/projects/$arc_id/repository/tree" > .tmp/02_current_arc_tree.json

  ### check that file `isa.investigation.xlsx` exists at ARC root
  ### if yes: download and dump
  ### if no: error message

  inv_path=$(grep -oE '"path":"isa.investigation.xlsx",' .tmp/02_current_arc_tree.json)

  ### check if variable is empty
  if [ -z "$inv_path" ]
  then
    printf "Missing 'isa.investigation.xlsx' at the root of $arc_path\n"
    printf "\n$arc_id\t$arc_path\tisa.investigation.xlsx missing" >> .tmp/02_investigations/arc_list.tsv
  else
```

```
    curl -L --silent --request GET --header "PRIVATE-TOKEN: $gitlab_pat
      " "https://git.nfdi4plants.org/api/v4/projects/$arc_id/
      repository/files/isa%2Einvestigation%2Exlsx/raw?ref=main" -o .
      tmp/02_investigations/$arc_id'_isa.inv.xlsx'
    printf "\n$arc_id\t$arc_path\tisa.investigation.xlsx detected" >> .
      tmp/02_investigations/arc_list.tsv
  fi

  rm .tmp/02_current_arc*

done
```

scripts/03_parse_isaInvxlsx.R

```

#############################################################
### Script to read metadata from an isa.investigation.xlsx
#############################################################

## rough idea:

# 0. Takes two arguments as CLI input: <ARC id> and <isa.investigation.
   xlsx>
# 1. Check whether its an investigation sheet or loop over sheets
# 2. focus on investigation only
# 3. subset into investigation sections
# 4. Store JSON-like as (nested) lists
#    - column 1 = keys
#    - column(s) 2:n = values as a list
# 5. put out as RData for further processing


########################
### Setup
########################

### If package "readxl" is not installed, install it.
# if(!require("readxl", quietly = TRUE)){install.packages("readxl")}

### load the package
library(readxl)


########################
### Inputs
########################

args = commandArgs(trailingOnly=TRUE)

```

```r
# test if arguments are supplied: if not, return an error
if (length(args)!=2) {

  stop("<ARC id> and <isa.investigation.xlsx> must be supplied as
      arguments", call.=FALSE)

  } else if (length(args)==2) {

  # default output file
  isa_inv_wb <- args[2]
  print(paste("Reading file", isa_inv_wb))
  arc_id <- args[1]
}

########################
### read data from excel
########################

### loop over sheets of workbook
for(sheet in excel_sheets(isa_inv_wb)){

  ### read sheet
  current_sheet <- as.data.frame(read_xlsx(isa_inv_wb, col_names = F,
      sheet = sheet, .name_repair = "minimal"))

  ### Simple sanity check for ISA investigation format

  if(current_sheet[1, 1] == "ONTOLOGY SOURCE REFERENCE" & current_sheet
      [2, 1] == "Term Source Name"){

    print(paste("Reading from excel sheet", sheet))
    invdata <- current_sheet

  }else{
    print(paste("Excel sheet", sheet, "is not in ISA investigation
        format"))
    invdata <- NULL
  }

}

### Stop if no proper ISA sheet detected
if(is.null(invdata)){stop(simpleError("No valid ISA investigation sheet
    detected"))}


########################
### wrangle / extract only relevant data
########################


```

```r
### subset to investigation only (excluding study, assay layers)
investigation_data <- invdata[grepl("^investigation",  invdata[, 1],
    ignore.case = T), ]

### first column as row names
rownames(investigation_data) <- investigation_data[,1]
investigation_data2 <- investigation_data[,-1, drop = F]

### extract investigation subsections (some redundancy with above)
inv_sections <- which(grepl("^INVESTIGATION",  row.names(investigation_
    data2), ignore.case = F))

investigation_list <- list()
for(i in 1:length(inv_sections))
{

  if(i == length(inv_sections))
    {
    section_range <- (inv_sections[i] + 1):nrow(investigation_data2)
    }else{
    section_range <- (inv_sections[i] + 1):(inv_sections[i+1] - 1)
    }

  current_section <- investigation_data2[section_range, , drop =F]

  ### remove columns that are only NA
  current_section <- current_section[, apply(current_section, 2,
      function(x){sum(is.na(x)) != nrow(current_section)}), drop = F]

  ### transpose / pivot data to transform into list
  current_section_transposed <- as.data.frame(t(current_section))
  rownames(current_section_transposed) <- NULL

  # TODO stupid work-around to circumvent the bug with a section having
     only NAs
  if(nrow(current_section_transposed) == 0){current_section_transposed
      [1, ] = NA}

  current_section_transposed$arc_id <- arc_id

  ### extract current section name
  current_section_name <- row.names(investigation_data2)[inv_sections[i
      ]]

  # ### transform to named list, omitting NAs
  # investigation_list[[current_section_name]] <- lapply(current_
      section_transposed, function(v){v[!is.na(v)]})
  #
  # stack(current_section_transposed)

  investigation_list[[current_section_name]] <- current_section_transposed

}

########################
### output to .RData
########################

if(!dir.exists(".tmp/03_rdata_dumps/")){dir.create(".tmp/03_rdata_dumps/")}

print(paste0("Storing outputs in: .tmp/03_rdata_dumps/", arc_id, ".Rdata"))

save(investigation_list, isa_inv_wb, arc_id, file = paste0(".tmp/03_rdata_dumps/", arc_id, ".RData"))
```

scripts/03_pull_together.R

```
############################################################
### Script to pull together output of previous scripts
############################################################

### for loop over available RData dumps from 03_parse_isaInvxlsx.R

all_arcs <- list()

for(i in dir(".tmp/03_rdata_dumps/", full.names = T, pattern = ".RData"))
{
  ### load the data
  load(i)

  ### store in named list

  all_arcs[[arc_id]] <- investigation_list

}

### row-bind the second-level (i.e. INVESTIGATION "sections") of the lists, respectively

all_arcs_db <- do.call(Map, c(f = rbind, all_arcs))

### read the arc_list (translating the ARC id to the ARC path) produced by 02_read_from_gitlab.sh
### and append to above list

```

```r
arc_list <- read.table(".tmp/02_investigations/arc_list.tsv", sep = "\t
    ", header = T)
colnames(arc_list)[1] <- "arc_id"

# all_arcs_db[["arc_list"]] <- arc_list

### store the output as RData

save(all_arcs, all_arcs_db, arc_list, file = ".tmp/03_allARCs.RData")
```

scripts/05_pull_together_sql.R

```r
#############################################################
### Convert data into SQLite database
#############################################################

# if(!require("DBI", quietly = TRUE)){install.packages("DBI")}
library(DBI)

load(".tmp/03_allARCs.RData")

### Write into an SQLite DB file

mydb <- dbConnect(RSQLite::SQLite(), "yourARCs_database.sqlite")

for(i in names(all_arcs_db))
{
  dbWriteTable(mydb, i, all_arcs_db[[i]], overwrite = T)
}

dbWriteTable(mydb, "ARC list", arc_list, overwrite = T)

dbListTables(mydb)
dbDisconnect(mydb)
```

scripts/04_searchApp/app.R

```r
#############################################################
### The ARC-Finder GUI Shiny App
#############################################################

suppressMessages(library(tidyverse))
suppressMessages(library(shiny))

# load(".tmp/03_allARCs.RData")

```

```
### Flatten ALL values into a 3-column (arc_id | key | value) df to
    provide search across "any field"

all_values <- do.call(
  rbind.data.frame, lapply(all_arcs_db, function(x){
  pivot_longer(x, cols = setdiff(colnames(x), "arc_id"), values_drop_na
      = T)}))

all_values <- as.data.frame(all_values)

arc_list <- unique(arc_list)

shinyApp(
    ui = pageWithSidebar(
        headerPanel("ARC Finder"),
        sidebarPanel(
            selectizeInput('search_key', 'Select a field to search',
                choices = c("Any field", unique(all_values$name))),
            uiOutput("search_field"),
            selectInput(inputId = "arc_path", label = "Select an ARC
                for details", choices = NULL),
            br(),
            actionButton("go", "Show this ARC"),
            h3("Access this ARC in the DataHUB"),
            uiOutput("arc_gitlab"),

        ),

        mainPanel(
          h3("Overview"),
          tableOutput("table_INV"),
          br(),
          h3("Publications"),
          tableOutput("table_INV_PUBS"),
          br(),
          h3("Contacts"),
          tableOutput("table_INV_Contacts")
        )
    ),

    server = function(input, output, session) {


      ##### reactive input field for text-search


      output$search_field <- renderUI({

        # check whether user wants to filter by cyl;
        # if not, then filter by selection
        if ('Any field' %in% input$search_key) {
```

```
       df <- all_values
     } else {
       df <- subset(all_values, name == input$search_key)
     }


    selectizeInput('search_value', 'Search the ARC database', choices = c("", sort(unique(df$value))))

    })


    ##### ARC choices (arc_id) matching user-input

     arc_choices_id <- reactive({

     if ('Any field' %in% input$search_key) {

      subset(all_values, value == input$search_value, arc_id, drop = T)

      } else {

      subset(all_values, name == input$search_key & value == input$search_value, arc_id, drop = T)
      }

     })


     ### retrieve path for matching ARCs from arc list

     arc_choices_path <- reactive({

     subset(arc_list, arc_id %in% arc_choices_id(), ARC.path, drop = T)

     })

    ##### reactive ARC selection: updated Input to let user pick from


     observe({
      updateSelectInput(session = session, inputId = "arc_path", choices = arc_choices_path())
      })


    #### User's ARC choice

     selected_arc <- eventReactive(input$go, {

        all_arcs[[as.character(subset(arc_list, ARC.path %in% input$arc
            _path, arc_id, drop = T))]]

        })


      ##### render table INVESTIGATION

        output$table_INV <- renderTable(colnames = F, rownames = F, {

          selected_table <- selected_arc()$INVESTIGATION
          selected_table <- selected_table[, -which(colnames(selected_
              table) == "arc_id")]

          selected_table$pivot_col <- row.names(selected_table)
          long <- pivot_longer(selected_table, setdiff(colnames(
              selected_table), "pivot_col"), values_drop_na = T)

          pivot_wider(long, names_from = pivot_col)

        })

      ##### render table INVESTIGATION PUBLICATIONS

        output$table_INV_PUBS <- renderTable(colnames = F, rownames = F
            , {

          selected_table <- selected_arc()$`INVESTIGATION PUBLICATIONS`
          selected_table <- selected_table[, -which(colnames(selected_
              table) == "arc_id")]

          selected_table$pivot_col <- row.names(selected_table)
          long <- pivot_longer(selected_table, setdiff(colnames(
              selected_table), "pivot_col"), values_drop_na = T)

          pivot_wider(long, names_from = pivot_col)


        })

      ##### render table INVESTIGATION CONTACTS

        output$table_INV_Contacts <- renderTable(colnames = F, rownames
            = F, {


          selected_table <- selected_arc()$`INVESTIGATION CONTACTS`
          selected_table <- selected_table[, -which(colnames(selected_
              table) == "arc_id")]

          selected_table$pivot_col <- row.names(selected_table)
          long <- pivot_longer(selected_table, setdiff(colnames(selected_table), "pivot_col"), values_drop_na = T)

          pivot_wider(long, names_from = pivot_col)

        })

    ##### render link to gitlab

        output$arc_gitlab <- renderUI(a(href = paste0('https://git.nfdi4plants.org/', input$arc_path),
                                        paste0('https://git.nfdi4plants.org/', input$arc_path) , target="_blank"))


        session$onSessionEnded(function() {
          stopApp()
        })
    }

)
```