



Fairspace is a secure place for managing research data. Research teams have their own workspaces in which they can manage research data collections. Researchers can upload directories and files to data collections. Data access is organised on data collection level. Collections can be shared with other teams or individual researchers. Also, collections can be published for all researchers in the organisation.

Collections and files can be annotated with descriptive metadata. The metadata is stored using the [Resource Description Framework \(RDF\)](#) in an [Apache Jena](#) database. For the metadata, a data model can be configured that suits the data management needs of the organisation. The data model is specified using the [Shapes Constraint Language \(SHACL\)](#), see the section on [Data model and view configuration](#). Descriptive metadata entities (e.g., subjects, projects, samples) should be added to the database by a careful process, ensuring that duplicates and inconsistencies are avoided and all entities have proper unique identifiers. The application provides overviews of the available metadata entities. In the collection browser, researchers can link their collections and file to these entities or add textual descriptions and key words.

Key features

- Fairspace is a data repository that enables researchers to securely **store** and **organise** their research data sets, and **share** the data with collaborators.
- Fairspace lets researchers annotate their data collections with relevant metadata properties and link the data to associated metadata entities (subjects, samples, projects, etc.). This helps researchers find their own data and make it **findable** for others, contributing to implementation of the [FAIR principles](#).
- Fairspace ensures that all metadata entities have a unique identifier and validates metadata consistency and validity upon data entry.
- Fairspace allows organisations to use **customise** the configured data model, by specifying custom entity types and constraints. This enables the adoption of community standards for metadata relevant for the research domain, which contributes to the **reusability** of the data.
- Fairspace uses the [Resource Description Framework \(RDF\)](#) and [WebDAV](#) standards for data exchange, and stimulates the uses of standard vocabularies, contributing to **interoperability** of data.

Table of contents

Usage	3
User interface	3
Interfaces for accessing and querying data (API)	8
Structure and terminology	12
Workflow and access modes	13
Roles and permissions	14
Data model and view configuration	16
Limitations	20
Controlled vocabularies	20
Installation and configuration	22
Local development	22
Kubernetes and helm	22
Design	25
Storage	25
License	26

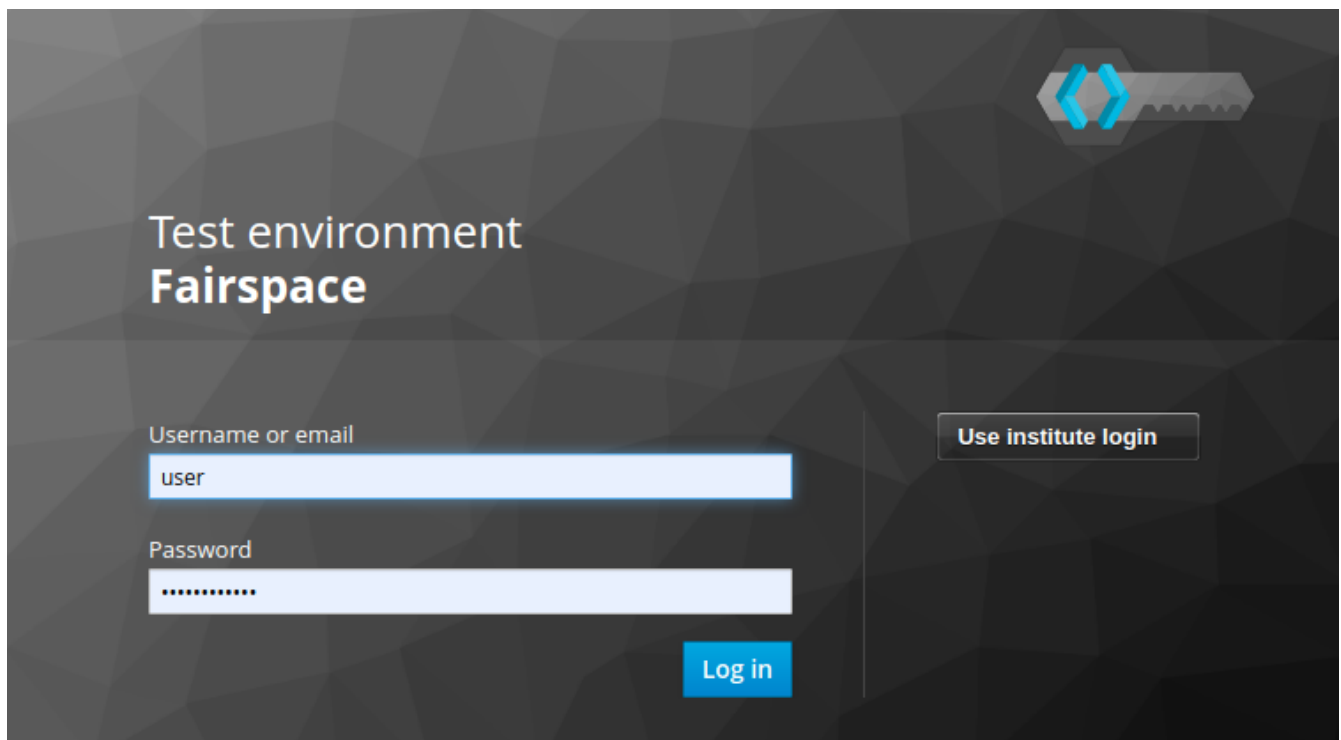
Usage

User interface

Login

Users are authenticated using [Keycloak](#), an open-source identity provider that provides secure authentication methods and can be configured to integrate with institutional identity providers using user federation or identity brokering, see the [Keycloak server administration](#) pages.

The user either logs in directly using Keycloak or is forwarded to a configured external login:

The image shows a login page for 'Test environment Fairspace'. The background is dark with a geometric pattern. In the top right corner, there is a logo consisting of a blue double arrow pointing right, followed by a grey key icon. The text 'Test environment' is in a light grey font, and 'Fairspace' is in a larger, bold white font. Below this, there are two input fields: 'Username or email' with the text 'user' entered, and 'Password' with masked characters '.....'. To the right of these fields is a button labeled 'Use institute login'. Below the password field is a blue 'Log in' button.

Workspaces

Users enter Fairspace on the workspaces page that lists all workspaces. A workspace represents a team in the organisation that collaborates on research data collections.

The screenshot displays the 'Workspaces' overview page. The table lists the following data:

Name ↑	Collections	Members	Managers
Fairspace development team Scripts and data of the Fairspace development team	3	2	First Organisation Admin
Test Test workspace	8	3	First Organisation Admin

At the bottom of the table, it indicates 'Rows per page: 10' and '1-2 of 2'.

Workspace administrators can edit the workspace overview page and manage workspace membership. All workspace members can add collections to the workspace.

The screenshot displays the 'Fairspace development team' workspace overview page. The page includes the following details:

- Workspace Name:** Fairspace development team
- Short description:** Scripts and data of the Fairspace development team
- Keywords:** dev

The page has tabs for OVERVIEW, USERS, and COLLECTIONS. The left sidebar contains links for Workspaces, Collections, Metadata, and JupyterHub. The top right shows the user YGRITTE.

Collections

The contents of collections can be navigated in the collections browser. It behaves like a regular file browser. Click to select a directory or file and see its metadata, double click to navigate into directories or open a file.

Access is managed on collection level. Users with at least write access to a collection can upload files or directories, rename or delete files, restore old file versions, and edit the associated metadata.

Collections / Performance tests / metrics / December 2020

Search [COLLECTION METADATA SEARCH](#) ☐ Show deleted

Name ↑	Size	Last modified
performance_metrics.csv	1.04 kB	19:03

Rows per page: 10 1-1 of 1

Performance tests

Performance metrics scripts and data

Owner workspace
Fairpace development team

Status
Active
Editing data and metadata enabled.

Write access
You can upload files and add metadata to this collection.

Metadata for Performance tests

Metadata for metrics

Metadata for metrics/December 2020

Metadata for metrics/December 2020/performance_metrics.csv

performance_metrics.csv

Description

Fairpace 0.7.4-SNAPSHOT

Metadata forms

Users with write access to the collection can annotate collections, directories and files using *metadata forms*. Free text fields, like description and key words, can be entered freely, links to shared entities, like subjects, samples and projects, or values from a controlled vocabulary, like taxonomy or analysis type, can be selected from a list:

Metadata for dir_1/samples.csv



samples.csv

Description

Measurement data

Keywords

2021

lab



Is about biological sample

ab2b



Sample f862313b-ab2b-4d19-bd0b-6a7c2c07c759

Sample 4b83486d-fdfd-43d4-ab2b-cb2728ebdee7



Type of analysis



Content type

text/csv

Created

10:23

Created by

Ygritte

UPDATE CANCEL

The shared metadata entities and controlled vocabularies cannot be added via the user interface. The [Metadata upload API](#) should be used for that instead.

Metadata upload

Another way to annotate directories and files is by uploading a comma-separated values (CSV) file with metadata. This section describes the CSV-based format used for bulk metadata uploads.

The file should be a valid CSV-file:

- Records are separated with a `,`-character.
- Values may be enclosed in double quotes: `"value"`.
- In values that contain a double, the double quotes need to be escaped by replacing them with double double quotes: Example `"quoted" text` becomes `"Example ""quoted"" text"`.

In the metadata upload, lines starting with `#` are ignored. These lines are considered to be comments.

The file should have a header row containing the names of the columns. The mandatory `Path` column is used for the file path. For the property columns, the name should match exactly the name of the property in the database.

The format of the values is as follows:

- *Path*: the relative path to a file or a directory (relative to the collection or directory where the file is uploaded). Use `./` for the current directory or collection.
- *Entity types* can be referenced by ID or unique label.
- Multiple values must be separated by the pipe symbol `|`, e.g., use `test|lab` to enter the values `test` and `lab`.

The file can be uploaded to the current directory by dropping the file in the metadata panel of the directory, or by selecting the metadata upload button.

By hovering over the metadata upload button, a link to a *metadata template file* becomes available:



The file describes the format in commented lines and contains the available properties in the header row.

Example 1. Example metadata file

An example comma-separated values file with metadata about the current directory `./`, which is annotated with a description and two key words (`sample` and `lab`), and the file `test.txt` which is linked to Subject 1 by the unique subject label and to the RNA-seq analysis type by the analysis type identifier (`06-12`).

```
Path,Is about subject,Type of analysis,Description,Keywords
./,,,Directory with samples,sample|lab,
test.txt,Subject 1,https://institut-curie.org/analysis#06-12,,
```

Metadata

Explore metadata and find associated collections and files.

The screenshot displays the YGRITTE Metadata interface. On the left, there is a sidebar with filters for 'Gender', 'Tumor topography', 'Morphology', 'Laterality', 'Event type', and 'Age at diagnosis'. The 'Tumor topography' filter is expanded, showing options like 'Pyiform sinus', 'RECTOSIGMOID JUNCTION', 'Stomach, NOS' (selected), 'TONSIL', 'Tonsillar fossa', 'Upper respiratory tract, NOS', and 'Ventricle, NOS'. The main panel shows a table of samples with columns: Sample, Tumor cellularity, Sample nature, and Gender. The table lists 12 samples, with the last one, 'Sample ff40f912-20e0-4c66-969e-f8cfbd115287', highlighted in pink. Below the table, it indicates 'Rows per page: 10' and '1-10 of 92'. On the right, a detailed view for the selected sample is shown, including its ID, sample nature ('Frozen Specimen'), gender ('Female'), and associated diagnosis and subject information.

Sample	Tumor cellularity	Sample nature	Gender
Sample 89bcab1f-9ae9-44de-bc74-0b4e6cd7fcc9	65	Protein	Male
Sample cb520ef0-4e05-432f-af7c-c7a4665bc3bb	47	Plasma	Male
Sample 98d92985-d3e9-4d42-8e10-ced35aab6908	37	Urine	Male
Sample 8103c2c8-bde7-40a2-86fa-ae9968d66376	54	Urine	Female
Sample 4bc348b6-01a6-4911-b744-3231aef45bc2	50	Blood	Male
Sample 1e456cb4-84a2-4570-9988-e7c67b8df9e9	60	Peripheral Blood Mononuclear Cell	Female
Sample ed1933af-dc14-454b-b3b7-f121a1531b6e	63	Plasma	Female
Sample 710f6e48-62cd-44fb-8f71-4997734c54b6	70	Plasma	Male
Sample 119623e6-5969-4203-8360-b2ad9a19d6c4	51	Plasma	Male
Sample ff40f912-20e0-4c66-969e-f8cfbd115287	38	Frozen Specimen	Female

Interfaces for accessing and querying data (API)

Authentication

OpenID Connect (OIDC) / OAuth2 workflow

Via header, via session.


```
import logging
import requests
import sys
import time

log = logging.getLogger()

def fetch_access_token(keycloak_url: str,
                      realm: str,
                      client_id: str,
                      client_secret: str,
                      username: str,
                      password: str) -> str:
    """
    Obtain access token from Keycloak
    :return: the access token as string.
    """
    params = {
        'client_id': client_id,
        'client_secret': client_secret,
        'username': username,
        'password': password,
        'grant_type': 'password'
    }
    headers = {
        'Content-type': 'application/x-www-form-urlencoded',
        'Accept': 'application/json'
    }
    response = requests.post(f'{keycloak_url}/auth/realms/{realm}/protocol/openid-connect/token',
                            data=params,
                            headers=headers)

    if not response.ok:
        log.error('Error fetching token!', response.json())
        sys.exit(1)
    data = response.json()
    token = data['access_token']
    log.info(f"Token obtained successfully. It will expire in {data['expires_in']} seconds")
    return token
```

Basic authentication

Use the `base64` encoded `username:password` in the `Authorization` header.

```
curl -v -H "Authorization: Basic $(echo -n "${USERNAME}:${PASSWORD}" | base64)"  
http://localhost:8080/api/users/current
```

Automatic authentication in Jupyter Hub

Metadata upload API

Metadata can be specified using:

- turtle
- json-ld

Example file: `testdata.ttl`:

```
@prefix example: <https://example.com/ontology#> .  
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .  
@prefix subject: <http://example.com/subjects#> .  
@prefix file: <http://example.com/api/webdav/> .  
@prefix gender: <http://hl7.org/fhir/administrative-gender#> .  
@prefix ncbitaxon: <https://bioportal.bioontology.org/ontologies/NCBITAXON/> .  
@prefix dcat: <http://www.w3.org/ns/dcat#> .  
  
subject:s1 a example:Subject ;  
           rdfs:label "Subject 1" ;  
           example:isOfSpecies ncbitaxon:9606 .  
  
file:coll1\coffee.jpg  
  dcat:keyword "fairspace", "java" ;  
  example:aboutSubject example:s1 .
```

Example with Python.

```
import logging
from requests import Session
import sys

log = logging.getLogger()

session = Session()
with open('testdata.ttl') as testdata:
    response: Response = session.put(f"{server_url}/api/metadata/",
                                     data=testdata.read(),
                                     headers={'Content-type': 'text/turtle'})
    if not response.ok:
        log.error('Error uploading metadata!')
        log.error(f'{response.status_code} {response.reason}')
        sys.exit(1)
```

```
curl -v
```

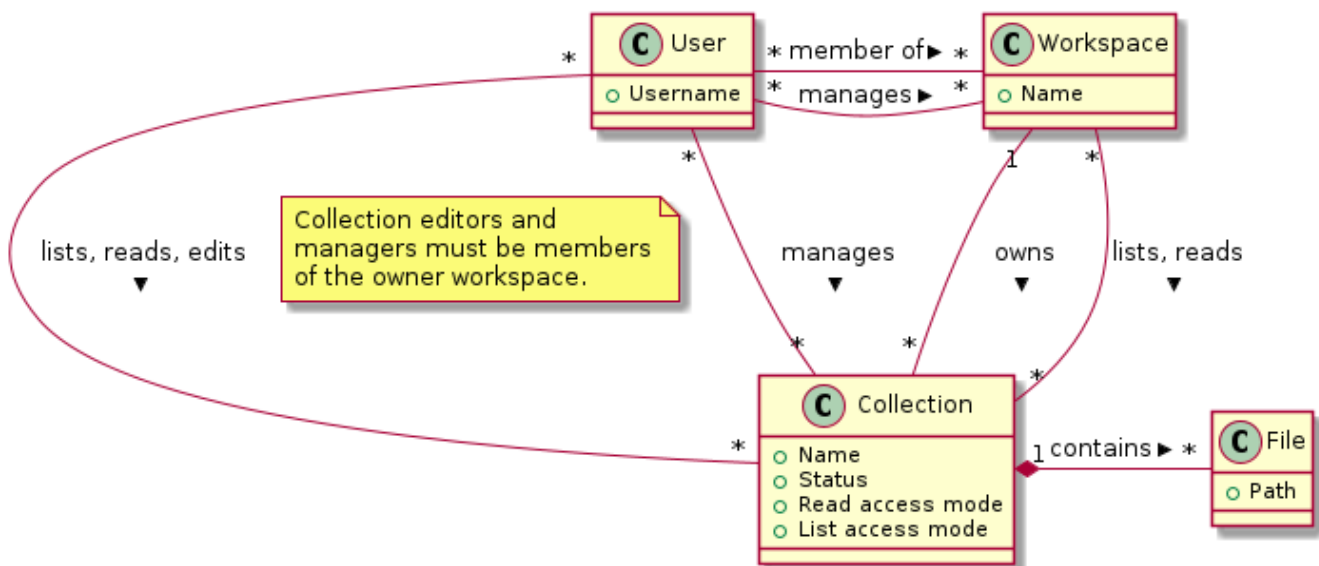
WebDAV

Structure and terminology

In this section we describe in detail the main concepts and components of the Fairspace data repository and how they relate to each other.

The core entities of the data repository are:

- *Users*: individual users in the organisation, looking for data, contributing to data collections or managing data.
- *Workspaces* (for projects, teams): entities in the system linked, representing a group of users, to organise data collections and data access.
- *Collections*: entities in the system to group data files. These are the minimal units of data for data access and data modification rules.
- *Files*: The smallest units of data that the system processes. Files always belong to a single collection. Files can be added, changed and deleted, but not in all collection states. Changing a file creates a new version. Access to a file is based on access to the collection the file belongs to. Files can be organised in *Directories*, which we will leave out of most descriptions for brevity.



The diagram above sketches the relevant entities and actors. The basic structure consists of users, workspaces, collections and files as represented in the system. Collections are the basic units of data access management. A collection is owned by a workspace. The responsibility for a collection is organised via the owner workspace: members of the owner workspace can be assigned as editors or managers of the collection. This reflects the situation where in an organisation, a data collection belongs to a project or a research team. This way the workspace represents the organisational unit that is responsible for a number of data collections (e.g., a research team or project). Data can be shared with other workspaces or individual users (for reading) and ownership may be transferred to another workspace (e.g., in the case the workspace is temporary, or when the organisation changes).

Fairspace provides a *data catalogue*, containing all the metadata, which is visible for all users with catalogue access (*View public metadata*). Users with metadata write access (*Add shared metadata*) can add metadata to the catalogue. Preferably this is done by an automated process that ensures

the consistency of the metadata and uniqueness of metadata entities. Metadata on collection and file level is protected by the access policy of the collections.

User administration is organised in an external component ([Keycloak]), but user permissions are stored in Fairspace. A back end application is responsible for storing the data and metadata, and for providing APIs for securely retrieving and adding data and metadata using standard data formats and protocols. A user interface application provides an interactive file manager and (meta)data browser and data entry forms based on the back end APIs. Besides the data storage and data management, Fairspace offers *analysis environments* using [Jupyter Hub](#). In Jupyter Hub, the data repository is accessible. Every user has a private working directory. We do no assumptions on the structure of the data or on the permissions of the external file systems that are connected to the data repository and referenced in the data catalogue. The organisation structure may be replicated in the different systems in incompatible ways, and the permissions may not be aligned.

Workflow and access modes

During the lifetime of a collection, different rules may be applicable for data modification and data access. In Fairspace, collections follow a workflow with the following statuses:

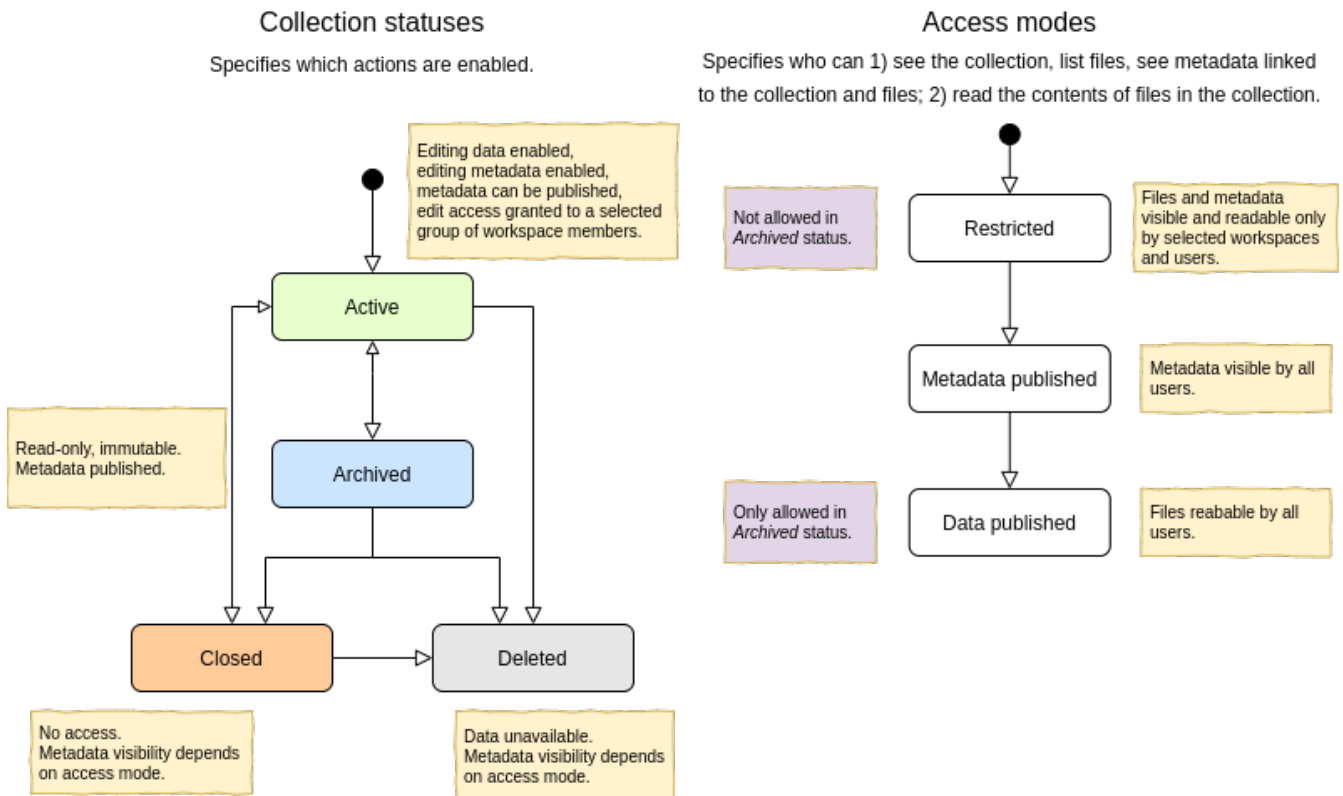
- *Active*: for the phase of data collection, data production and data processing;
- *Archived*: for when the data set is complete and is available for reuse;
- *Closed*: for when the data set should not be available for reading, but still needs to be preserved;
- *Deleted*: for when the data set needs to be permanently made unavailable. This status is irreversible. There is one exception to this rule – for the sake of data loss prevention, in special cases, administrators can still undelete a collection that was already deleted.

In these different statuses, different actions on the data are enabled or disabled. Also, visibility of the data and linked metadata depends partly on the collection status. We also distinguish three access modes for reading and listing files in a collection (where listing also includes seeing the metadata):

- *Restricted*: only access to explicitly selected workspaces and users;
- *Metadata published*: the collection and its files are visible, metadata linked to them is visible for all users;
- *Data published*: the files in the collection are readable for all users. This mode is irreversible. There is one exception to this rule – there might be a special situation, resulting from, e.g., a legal reason, when a collection has to be unpublished. This action is available to administrators, but it is highly discouraged, since the collection (meta)data may already be referenced in other systems.

The statuses and access modes, and the transitions between them are shown in the following diagram.

Collection editing and publication workflow



Roles and permissions

We distinguish the following roles in the solution:

- *User*: regular users can only view their own workspaces and collections.
- *View public metadata*: the user can view public metadata, workspaces, collections and files;
- *View public data*: the user can read public files;
- *Admin*: can create workspaces, assign roles and permissions;
- *Add shared metadata*: can add, modify and delete shared metadata entities.

Most users should have the *View public data* role. Only when the shared metadata may contain sensitive information that should not be visible for some users, the public data and public metadata roles should be discarded for those users.

Workspaces are used to organise collections in a hierarchy. On workspace level there are two access levels:

- *Manager*: can edit workspace details, manage workspace access and manage access to all collections that belong to the workspace;
- *Member*: can create a collection in the workspace.

Access to collections and files is managed on collection level. We distinguish the following access levels on collections:

- *List*: see collection, directory and file names and metadata properties/relations (only applicable

for collections shared via the *Metadata published* access mode);

- *Read*: read file contents;
- *Write*: add files, add new file versions, mark files as deleted;
- *Manage*: grant, revoke access to the collection, change collection status and modes.

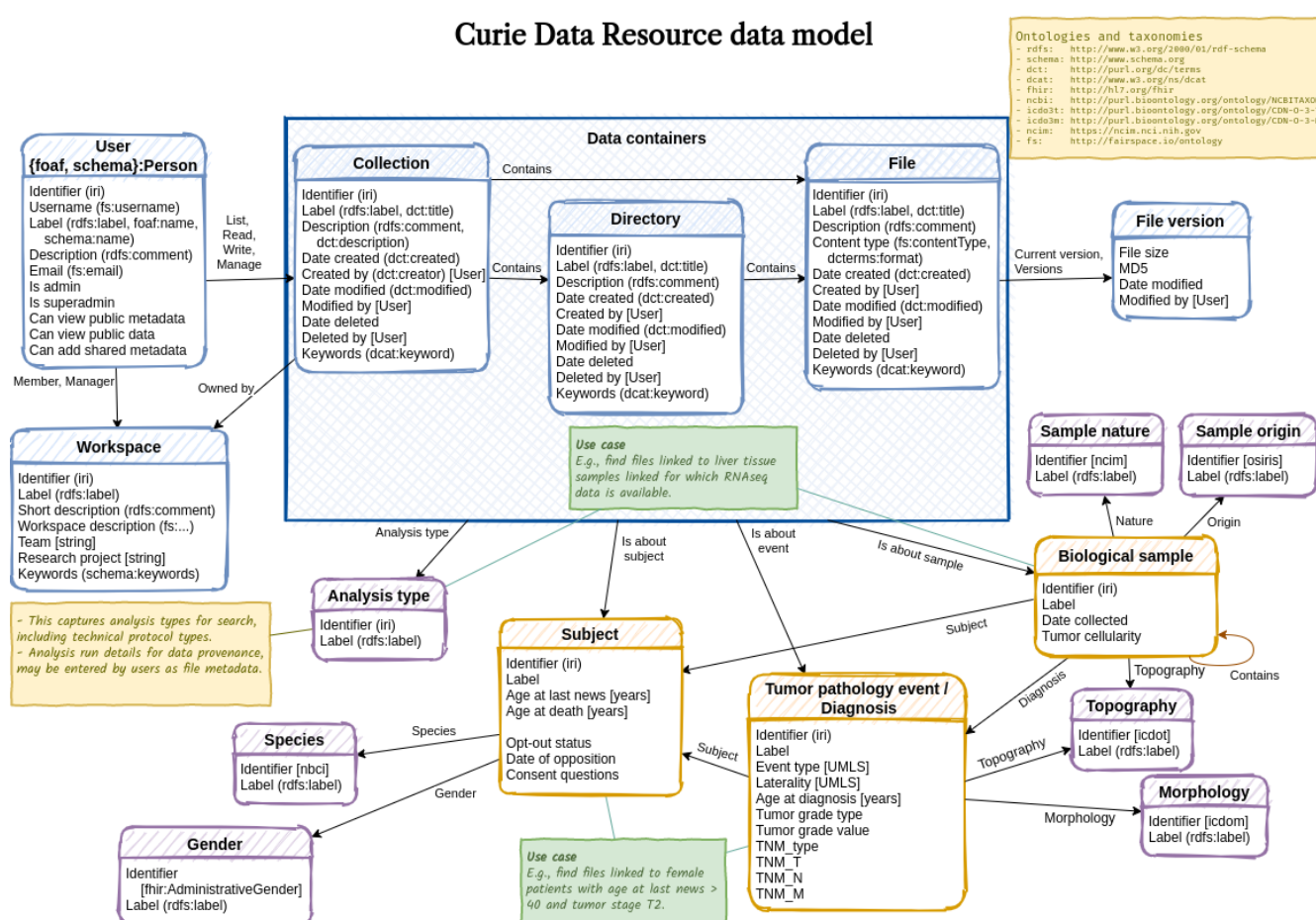
Access levels are hierarchical: the *Read* level includes the *List* level; the *Edit* level includes *Read* level; the *Manage* level includes *Edit* and *Read* level access. The user that creates the collection gets *Manage* access.

Data model and view configuration

FairSpace uses an [Apache Jena](#) database to store system metadata and the custom domain specific metadata. The data models for these metadata are defined using the [Shapes Constraint Language \(SHACL\)](#).

- The system metadata includes workspaces, collections, directories, files, file versions, users and access rights. The system data model is defined in [system-vocabulary.ttl](#)
- The customisable data model includes the custom (shared) metadata entities, custom controlled vocabulary types, and custom properties of the system entities. The default custom data model is defined in [vocabulary.ttl](#). This data model can be overridden by a data more suitable for your organisation.

A schematic overview of the default data model in [vocabulary.ttl](#):



The data model defines an entity-relationship model, specifying the entity types that are relevant to describe your data assets, the properties of the entities, and the relationships between entities.

In this example data model, the following custom entity types are defined:

- `example:Gender` with property *Label*;
- `example:Species` with property *Label*;
- `example:Subject` with properties *Gender*, *Species*, *Age at last news* and *Files*.

The system class `fs:File` is extended with the *Is about subject* property.

```
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix sh: <http://www.w3.org/ns/shacl#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix dash: <http://datashapes.org/dash#> .
@prefix fs: <https://fairspace.nl/ontology#> .
@prefix example: <https://example.com/ontology#> .

example:Gender a rdfs:Class, sh:NodeShape ;
  sh:closed false ;
  sh:description "The gender of the subject." ;
  sh:name "Gender" ;
  sh:ignoredProperties ( rdf:type owl:sameAs ) ;
  sh:property
  [
    sh:name "Label" ;
    sh:description "Unique gender label." ;
    sh:datatype xsd:string ;
    sh:maxCount 1 ;
    dash:singleLine true ;
    fs:importantProperty true ;
    sh:path rdfs:label
  ] .

example:Species a rdfs:Class, sh:NodeShape ;
  sh:closed false ;
  sh:description "The species of the subject." ;
  sh:name "Species" ;
  sh:ignoredProperties ( rdf:type owl:sameAs ) ;
  sh:property
  [
    sh:name "Label" ;
    sh:description "Unique species label." ;
    sh:datatype xsd:string ;
    sh:maxCount 1 ;
    dash:singleLine true ;
    fs:importantProperty true ;
    sh:path rdfs:label
  ] .
```

```

example:isOfGender a rdf:Property .
example:isOfSpecies a rdf:Property .
example:ageAtLastNews a rdf:Property .

example:Subject a rdfs:Class, sh:NodeShape ;
  sh:closed false ;
  sh:description "A subject of research." ;
  sh:name "Subject" ;
  sh:ignoredProperties ( rdf:type owl:sameAs ) ;
  sh:property
  [
    sh:name "Label" ;
    sh:description "Unique subject label." ;
    sh:datatype xsd:string ;
    sh:maxCount 1 ;
    dash:singleLine true ;
    fs:importantProperty true ;
    sh:path rdfs:label;
    sh:order 0
  ],
  [
    sh:name "Gender" ;
    sh:description "The gender of the subject." ;
    sh:maxCount 1 ;
    sh:class example:Gender ;
    sh:path example:isOfGender
  ],
  [
    sh:name "Species" ;
    sh:description "The species of the subject." ;
    sh:maxCount 1 ;
    sh:class example:Species ;
    sh:path example:isOfSpecies
  ],
  [
    sh:name "Age at last news" ;
    sh:description "The age at last news." ;
    sh:datatype xsd:integer ;
    sh:maxCount 1 ;
    sh:path example:ageAtLastNews
  ],
  [
    sh:name "Files" ;
    sh:description "Linked files" ;
    sh:path [sh:inversePath example:aboutSubject];
  ] .

```

```

example:aboutSubject a rdf:Property .

```

```

# Augmented system class shapes

```

```

fs:File sh:property
[
  sh:name "Is about subject" ;
  sh:description "Subjects that are featured in this collection." ;
  sh:class example:Subject ;
  sh:path example:aboutSubject
] .

```

All entity types have a unique label, specified using the `rdfs:label` predicate. The *Gender* and *Species* properties link the subject to an entity from the respective controlled vocabularies. The *Age at last news* property is a numerical (integer) value property.

The *Files* property of the *Subject* entity type is an example of an inverse relation. The link is defined on the file, but the link will be visible on the subject as well, because of this inverse relation.

The following guidelines should be followed when creating a custom data model.

- Define a namespace for your custom entities and properties, like `@prefix example: https://example.com/ontology# .` in the example.
- Each custom entity type must have types `rdfs:Class` and `sh:NodeShape`, the properties `sh:closed false` and `sh:ignoredProperties (rdf:type owl:sameAs)`, and a valid value for `sh:name`. The `sh:description` property is optional.
- Controlled vocabulary or terminology types are modelled as entity types as well, having only the *Label* (`rdfs:label`) property, see `example:Gender` and `example:Species`.
- Properties are specified using the `sh:property` property.
 - Every entity type must have a property *Label* (`sh:path rdfs:label`) of data type `xsd:string`. The label of an entity must be unique for that type. The label property should be singleton and marked `fs:importantProperty true`. If there are multiple properties, the label should have `sh:order: 0`.
 - Properties must have a valid value for `sh:name`. The `sh:description` property is optional.
 - A property must either have a `sh:datatype` property, specifying one of `xsd:string`, `xsd:integer` or `xsd:date`, or a property `sh:class` specifying an entity type as the target of a relationship.
 - The predicate used for the property (the middle part of the RDF triple) is specified with the `sh:path` property, e.g., `example:aboutSubject` for the *Is about subject* relation.
 - If a relationship is bidirectional, the path of the inverse relation is specified using `sh:inversePath`, see the *Files* property on the *Subject* entity type.
 - A property can be marked *mandatory* by specifying `sh:minCount 1`. A property can be marked *singleton* by specifying `sh:maxCount 1`.
 - A text property (with `sh:datatype xsd:string`) can be limited to a single line text field using `dash:singleLine true`.

Limitations

Although assigning multiple types to an entity is easy in RDF, Fairspace assumes entities to have a single type.

Inheritance is possible in SHACL, but not supported by Fairspace. Instead of specifying an entity type as a subtype of another, a single type can be specified with a *type* property, indicating the sub type of the entity.

E.g., instead of defining entity types *DNaseqAssay* and *RNaseqAssay* as sub types of *Assay*, a property type *assayType* can be defined on *Assay*, using a controlled vocabulary type *AssayType* with the assay types as values.

Controlled vocabularies

For controlled vocabulary types, e.g., *Gender* and *Species* in the example, you should insert the allowed values in the database by uploading a taxonomies file using the [Metadata upload API](#). An example taxonomy is in [taxonomies.ttl](#).

It is preferred to use existing standard taxonomies and labels. If that is not possible, please define your own namespaces for your custom taxonomies.

Example 3. Example taxonomy

In this example we use existing standard ontologies for the *Gender* and *Species* controlled vocabulary types.

- The [HL7 FHIR AdministrativeGender code system](http://hl7.org/fhir/administrative-gender#) for *Gender*.
- The [NCBI Organismal Classification](https://bioportal.bioontology.org/ontologies/NCBITAXON/) for *Species*.

```
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix example: <https://example.com/ontology#> .
@prefix gender: <http://hl7.org/fhir/administrative-gender#> .
@prefix ncbitaxon: <https://bioportal.bioontology.org/ontologies/NCBITAXON/> .

gender:male a example:Gender ;
  rdfs:label "Male" .
gender:female a example:Gender ;
  rdfs:label "Female" .

ncbitaxon:562 a example:Species ;
  rdfs:label "Escherichia coli" .
ncbitaxon:1423 a example:Species ;
  rdfs:label "Bacillus subtilis" .
ncbitaxon:4896 a example:Species ;
  rdfs:label "Schizosaccharomyces pombe" .
ncbitaxon:4932 a example:Species ;
  rdfs:label "Saccharomyces cerevisiae" .
ncbitaxon:6239 a example:Species ;
  rdfs:label "Caenorhabditis elegans" .
ncbitaxon:7227 a example:Species ;
  rdfs:label "Drosophila melanogaster" .
ncbitaxon:7955 a example:Species ;
  rdfs:label "Zebrafish" .
ncbitaxon:8355 a example:Species ;
  rdfs:label "Xenopus laevis" .
ncbitaxon:9606 a example:Species ;
  rdfs:label "Homo sapiens" .
ncbitaxon:10090 a example:Species ;
  rdfs:label "Mus musculus" .
```

Installation and configuration

Local development

Requires:

- yarn
- docker
- Java 15

To run the development version, checkout this repository, navigate to `projects/mercury` and run

```
yarn dev
```

If on MacOS, configure docker logging.... TODO As env variable, or in `.env` file: `DOCKER_LOGGING_DRIVER=json-file`.

This will start a Keycloak instance for authentication at port `5100`, the backend application named Saturn at port `8080` and the user interface at port `3000`.

At first run, you need to configure the service account in Keycloak.

- Navigate to <http://localhost:5100>
- Login with credentials `keycloak`, `keycloak`
- Grant `realm-management` roles in the Fairspace realm: `view-realm`, `manage-realm`, `manage-authorization`, `manage-users`.

Now everything should be ready to start using Fairspace:

- Navigate to <http://localhost:3000> to open the application.
- Login with one of the following credentials:

Username	Password
organisation-admin	fairspace123
user	fairspace123

Kubernetes and helm

You can deploy Fairspace on a Kubernetes cluster using [Helm](#). Helm charts for Fairspace are published to the public helm repository at <https://storage.googleapis.com/fairspace-helm>.

Instructions for deploying to Google Cloud

Download and install helm and gcloud

- Download **helm 2.14.3** from <https://github.com/helm/helm/releases/tag/v2.14.3>
- Extract the downloaded archive to `~/bin/helm` and check with:

```
~/bin/helm/helm version
```

- Install **kubect**l.
- Download and install the **Google Cloud SDK** (requires Python).
- Obtain credentials for Kubernetes:

```
gcloud container clusters get-credentials <cluster id> --zone europe-west1-b
```

Use **fairspacecluster** as cluster id for the CI environment. Ensure that your Google account has access to the **fairspace-207108** GCP project and log in using

```
gcloud auth login
```

- Check if all tools are correctly installed:

```
# List available clusters
gcloud container clusters list
# List Kubernetes namespaces
kubectl get ns
# List helm releases (deployments)
~/bin/helm/helm list
```

Initialise helm and add fairspace repository

```
# Initialise helm
~/bin/helm/helm init --client-only --stable-repo-url https://charts.helm.sh/stable
# Add the fairspace repo for reading
~/bin/helm/helm repo add fairspace https://storage.googleapis.com/fairspace-helm
# (Optional) Add the fairspace repo via the GCS plugin for writing
~/bin/helm/helm plugin install https://github.com/hayorov/helm-gcs.git --version 0.2.2
gcloud iam service-accounts keys create credentials.json --iam-account fairspace-207108@appspot.gserviceaccount.com
export GOOGLE_APPLICATION_CREDENTIALS=/path/to/credentials.json
~/bin/helm/helm repo add fairspace-gcs gs://fairspace-helm
```

Fetch chart

```
# Update repo
~/bin/helm/helm repo update
# Fetch the fairspace chart
~/bin/helm/helm fetch fairspace/fairspace --version 0.7.5
```

Deploy Fairspace

Create a new Kubernetes namespace:

```
kubectl create namespace fairspace-new
```

Create a new deployment (called *release* in helm terminology) and install the Fairspace chart:

```
~/bin/helm/helm install fairspace/fairspace --version 0.7.5 --name fairspace-new
--namespace=fairspace-new \
-f /path/to/values.yaml --set-file saturn.vocabulary=/path/to/vocabulary.ttl --set
-file saturn.views=/path/to/views.yaml
```

You can pass values files with `-f` and provide a file for a specified value with `--set-file`.

Update an existing deployment

To update a deployment using a new chart:

```
~/bin/helm/helm upgrade fairspace-new fairspace-0.7.5.tgz
```

With `helm upgrade` you can also pass new values files with `-f` and pass files with `--set-file` as for `helm install`.

Clean up deployment

To clean up an environment or completely reinstall an environment, you can use `helm del`.
:warning: Be careful, you may lose data!

```
~/bin/helm/helm del --purge fairspace-test
```


Design

Storage

RDF database using [Apache Jena](#) for:

- File metadata
- Permissions
- User metadata

File system data stored as blocks on the file system in append-only fashion.

License

...