Digital Milliet Documentation

Release 1.0

Bridget Almas, Anna Krohn, Marie-Claire Beaulieu

Dec 07, 2018

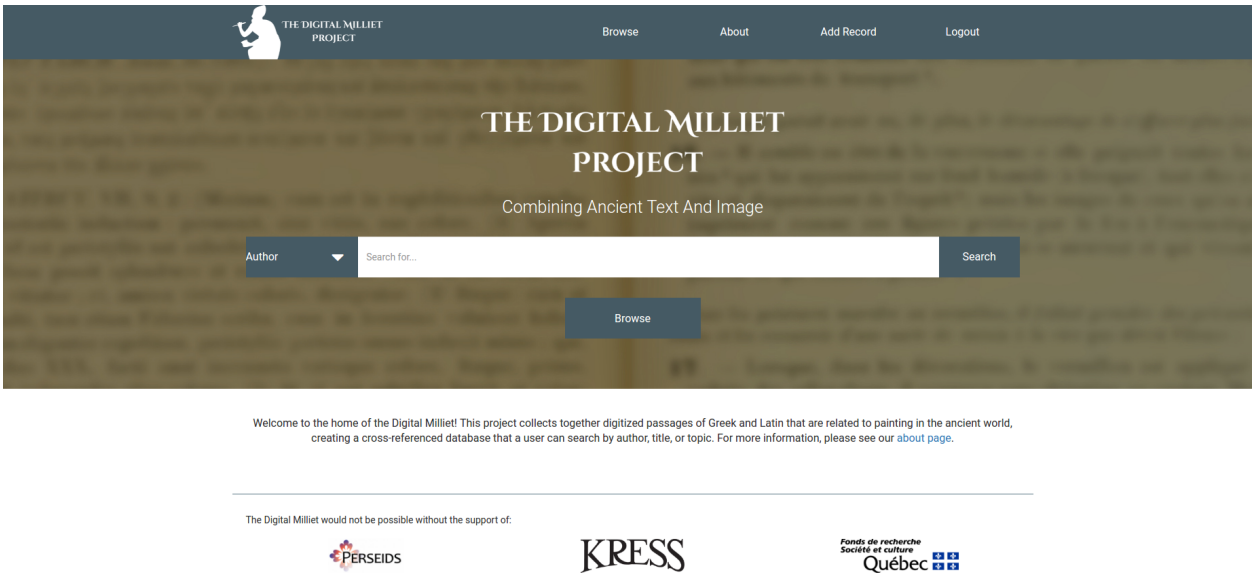
Contents:

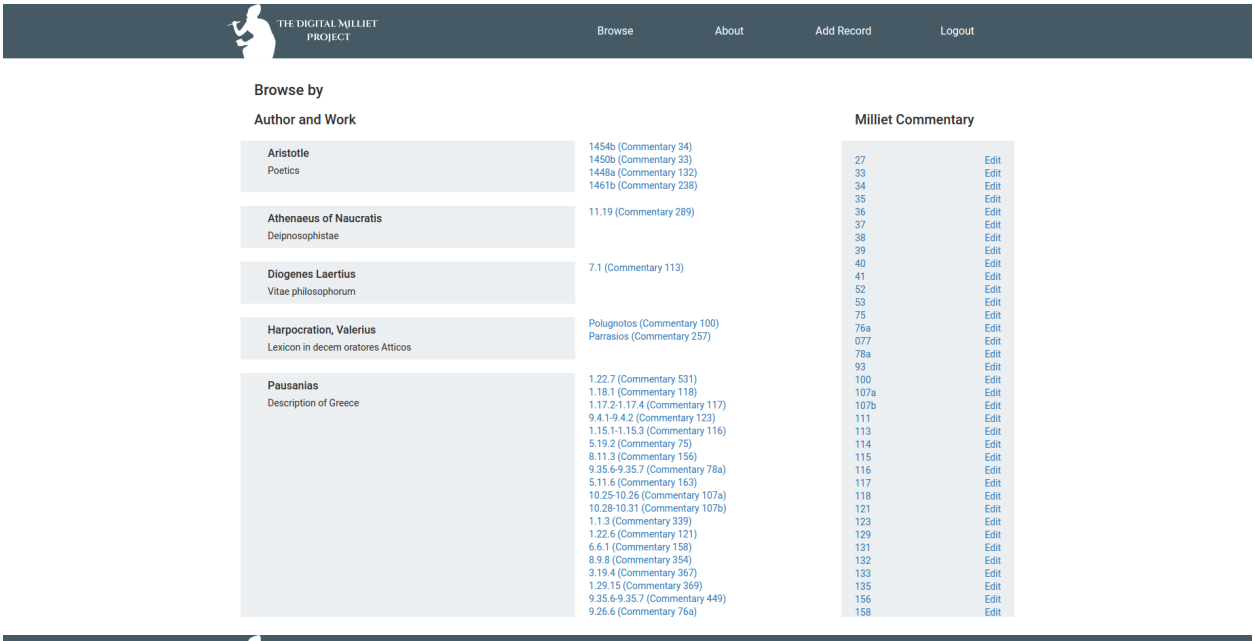
Full Documentation at http://digital-milliet.readthedocs.io/en/latest/

CHAPTER 1

Overview

The Digitat Milliet supports the creation and display of an interactive collection of ancient Greek and Latin texts about painting. It is a digital interpretation of “The Recueil des textes grecs et latins relatifs à la peinture ancienne” (“Collection of Greek and Latin Texts Concerning Ancient Painting”), the initiative of a French academic painter, Paul Milliet, who had a passion for ancient Greek culture.





The Digital Milliet is implmented as a Flask Application, backed by a MongoDB database, supported by external web services.

1.1 Installation Instructions

The following instructions are for setting up a Development environment for Digital Milliet.

Install Prerequisites:

- mongodb
- python 3.5, pip and virtualenv

```
sudo apt-get install -y python3-pip python3-dev build-essential mongo
```

Clone the repository

```
git clone https://github.com/perseids-project/digital_milliet
```

Setup the sample data

```
mongorestore digital_milliet/db/sample
```

Create a virtual environment

```
cd digital_milliet
virtualenv -p /path/to/python3 venv
source venv/bin/activate
python setup.py install
```

Run the code, installing test fixtures and with a fixed user:

```
python runtest.py --install --loggedin
```

Or with Docker and Docker Compose

```
git clone https://github.com/perseids-project/digital_milliet
cd digital_milliet
docker-compose build
docker-compose up
```

For production deployment, see Puppet manifests in the puppet subdirectory of this repository.

1.2 Configuration

All deployment specific variables and dependencies are specified in an external configuration file. By default the application looks for a configuration file named *config.cfg* in the digital_milliet base directory. An alternate path can be supplied in an argument to the DigitalMilliet Flask Application:

```
DigitalMilliet(app, config_files=["path/to/your/config.cfg"])
```

The default contents of this configuration file, with explanation of each setting, is provided below:

```
# Name of the Mongo database
MONGO_DBNAME = 'app'

# Secret key for Flask session
SECRET_KEY = 'development is fun'

# Perseids OAUTH Setup
# OAUTH_CONSUMER_KEY and OAUTH_CONSUMER_SECRET must be supplied by Perseids␣
↪Administrator for Production use
OAUTH_NAME = "digitalmilliet"
OAUTH_CONSUMER_KEY = 'dummy'
OAUTH_CONSUMER_SECRET = 'dummy'
OAUTH_REQUEST_TOKEN_PARAMS = {'scope': 'read'}
OAUTH_BASE_URL = 'https://sosol.perseids.org/sosol/api/v1/'
OAUTH_ACCESS_TOKEN_URL = 'https://sosol.perseids.org/sosol/oauth/token'
```

(continues on next page)

(continued from previous page)

```
OAUTH_ACCESS_TOKEN_METHOD = "POST"
OAUTH_REQUEST_TOKEN_URL = None
OAUTH_AUTHORIZE_URL = 'https://sosol.perseids.org/sosol/oauth/authorize'
OAUTH_CALLBACK_URL = 'https://digmill.perseids.org/digmil/oauth/authorized'

# Name of the collection for author records (future proofing to enable move to a␣
↪separate collection)
AUTHORS_COLLECTION = "annotation"

# Set this to the ID for the Perseids community id in which membership enables␣
↪Digital Milliet editorial permissions
ENFORCE_COMMUNITY_ID = None

# Not to be used in Production: eases development without OAuth Setup
OAUTH_USER_OVERRIDE = { 'oauth_user_uri' : 'http://sampleuseruri', 'oauth_user_name':
↪'Sample User' }

# Perseus Catalog API - Used for Lookup of Author and Work Metadata
CATALOG_API_URL = 'http://catalog.perseus.org/cite-collections/api'
CITE_URI_PREFIX = 'http://perseids.org/collections/'
CITE_COLLECTION = 'urn:cite:perseus:digmil'

# CTS API Endpoint for Retrieval of Primary Source Texts and Translations
CTS_BROWSE_URL = 'https://cts.perseids.org'
CTS_API_URL = 'https://cts.perseids.org/api/cts/'
CTS_API_VERSION = 5
```

1.3 Authentication and Authorization

The Digital Milliet application itself does not provide a user model or any AAI functionality.

The Create, Update and Delete functionality of the Digital Milliet application can be protected by the OAuth2 protocol. The location of the OAuth2 endpoint and other details must be supplied in these configuration settings:

```
OAUTH_NAME = "digitalmilliet"
OAUTH_CONSUMER_KEY = ''
OAUTH_CONSUMER_SECRET =''
OAUTH_REQUEST_TOKEN_PARAMS = {'scope': 'read'}
OAUTH_BASE_URL = ''
OAUTH_ACCESS_TOKEN_URL = ''
OAUTH_ACCESS_TOKEN_METHOD = "POST"
OAUTH_REQUEST_TOKEN_URL = None
OAUTH_AUTHORIZE_URL = ''
OAUTH_CALLBACK_URL = '<digmill_application_host>/oauth/authorized'
```

The deployment at https://digmill.perseids.org uses Perseids (https://sosol.perseids.org/sosol) as its OAuth2 provider. Perseids in turn delegates to Social Identity providers for user authentication. Perseids assigns a URI identifier to authenticated users and users supply a public-facing full name that they wish to be affiliated with their Perseids account. This information (the Perseids User URI and Full Name) are added as the creator associated with annotations created in the Digital Milliet application. Once a record is created, if it's edited by a user other than the creator, that user is added as an additional editor in the updated annotations.

Although not recommended for production use, it is possible to disable the OAuth2 protection by setting the name and URI to associate with all records via the *OAUTH_USER_OVERRIDE* configuration setting. This could be used in combination with a simpler authentication method such as HTTP Basic Authorization.

OAuth2 provides Authentication but not Authorization support. (By Authorization we mean restricting create/update/delete access of Digital Milliet entries to only specific authenticated users.) Implementing a full user model and role-based authorization was out of scope for development of the Digital Milliet application. A potential future goal is to use the Perseids platform to provide editorial review board functionality, removing the ability to edit annotations directly in the Digital Milliet application.

With this goal in mind, we implemented a Perseids-specific stop-gap solution to provide Authorization functionality to the Digital Milliet application. The application configuration allows for the specification of the identifier of a Perseids review community (via the *ENFORCE_COMMUNITY_ID* setting). If this is specified, then authenticated users must be a member of the Perseids Community with that id in order to be able to create, edit or delete entries in the Digital Milliet. If the *ENFORCE_COMMUNITY_ID* setting is left empty, this functionality is disabled and all authenticated users can create, edit or delete entries.

1.4 Design: Motivation, Standards, Dependencies

The aim behind the design of the application was to support the representation of each entry in the original "Recueil" as a graph of annotations.

The primary annotation of a Digital Milliet graph/record set is a Commentary targeting a stable CTS URN identifier of the primary source Greek or Latin text which was the subject of the entry in the "Receuil". This commentary annotation gets assigned an identifier which includes the original number of the entry in the "Receui". Throughout the code and interface, this is referred to as the "Milliet Number".

Additional annotations in each graph include a Bibliography, French and English translations of the primary source text, tags (freeform and semantic) as well as images representing the described artwork or related material. The images can also be annotated.

Entries are indexed for browsing both by Milliet Number and Author/Work/Passage of the target primary source text passage.

The Digital Milliet application retrieves Author and Work metadata for each primary source text is from the Perseus Catalog (http://catalog.perseus.org/).

We have used a non-standard form of a CITE URN to assign identifiers to each individual annotation in the graph. This may eventually be replaced by UUIDs or other identifier system.

In order to facilitate data reuse and interoperability we represent these annotations according to the Open Annotation data model (http://www.openannotation.org/), a standard data model for serializing annotations on resources in the world wide web. (This model has now evolved into the W3C Web Annotation Model). Image annotations adhere to the IIIF standard (http://iiif.io).

The original design called for primary source texts and translations to be identified only by their CTS URN identifiers and all textual passages retrieved at runtime from CTS Repositories.

However, as many of the texts and/or translations we need to refer to are not yet available online at a published CTS API endpoint, and the stability and long term sustainability of such end points are not clear, the application design was changed to enabled textual content to be included in addition to or instead of the CTS URN identifier of a text or translation.

The Digital Milliet application depends upon components of the CapiTainS suite (https://github.com/capitains) for its interaction with CTS endpoints and validation of CTS URN syntax.

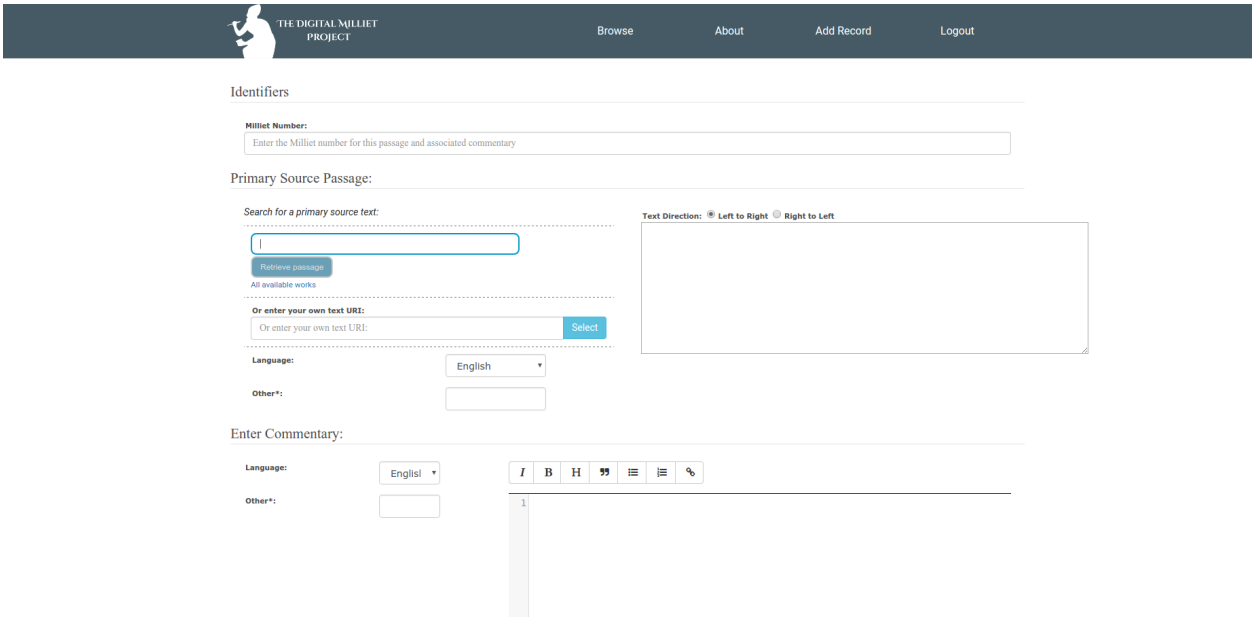
The application uses the IIIF standard for image referencing and annotations and reuses the open source Mirador Viewer (http://projectmirador.org/) to provide image display and annotation functionality.

1.5 Workflow

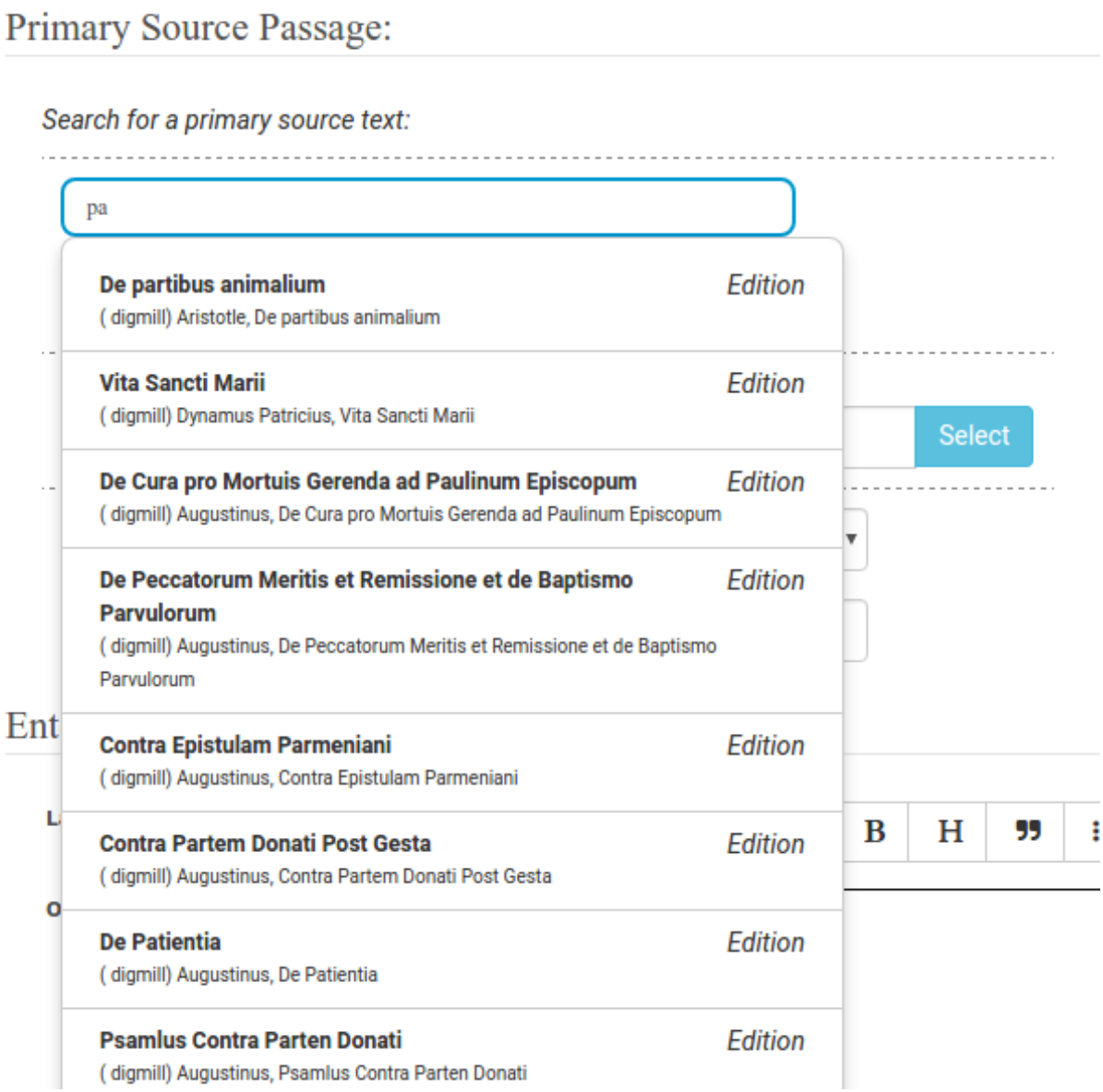
The primary workflow for creating a new entry in the Digital Milliet is described in the diagram below.

Individual components of an entry can also be edited or added separately after the initial data entry, via the Edit interface.

To create a new entry, you click the Add Record button to bring up the Create form:



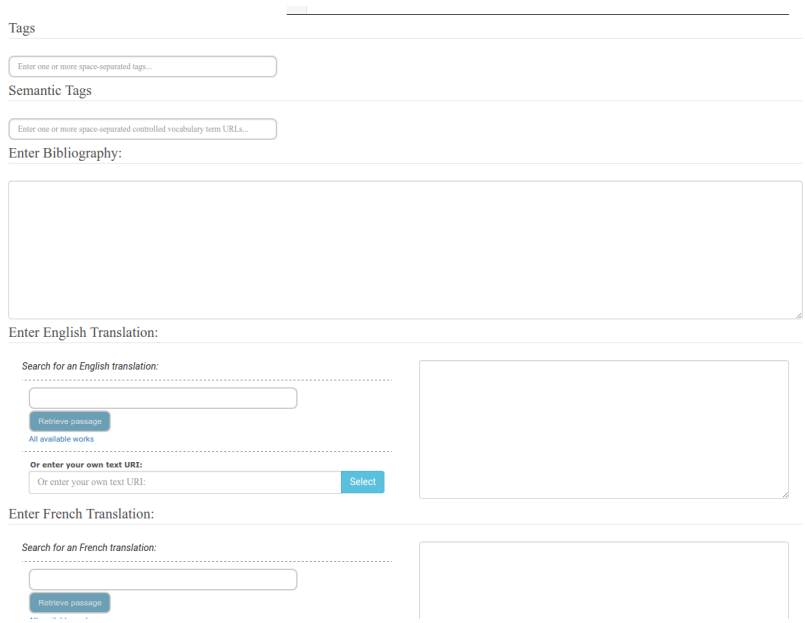
Use the typeahead features in the 'Search for a Primary Source Passage' to search for an existing text in the CTS Repository



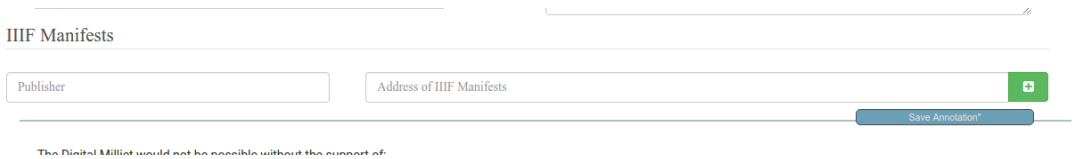
If found, you can enter the passage range you are interested in and then click 'Retrieve' to to retrieve the text.

If text you need is not found you can supply the text yourself in the input box.

Proceed to enter commentary text, tags and bibliography. Follow the same procedure for translations as you did for the primary source text.



If an image you want to associate with the entry is available in from an IIIF-compliant image server you can enter the publisher and URL of the IIIF manifest. This can be an image manifest, or a canvas manifest.



To edit an existing entry, you click the Edit button next to the Digital Milliet number on the Browse display. You must be logged into see this option.

Editing proceeds similarly to the process for creating a new entry.

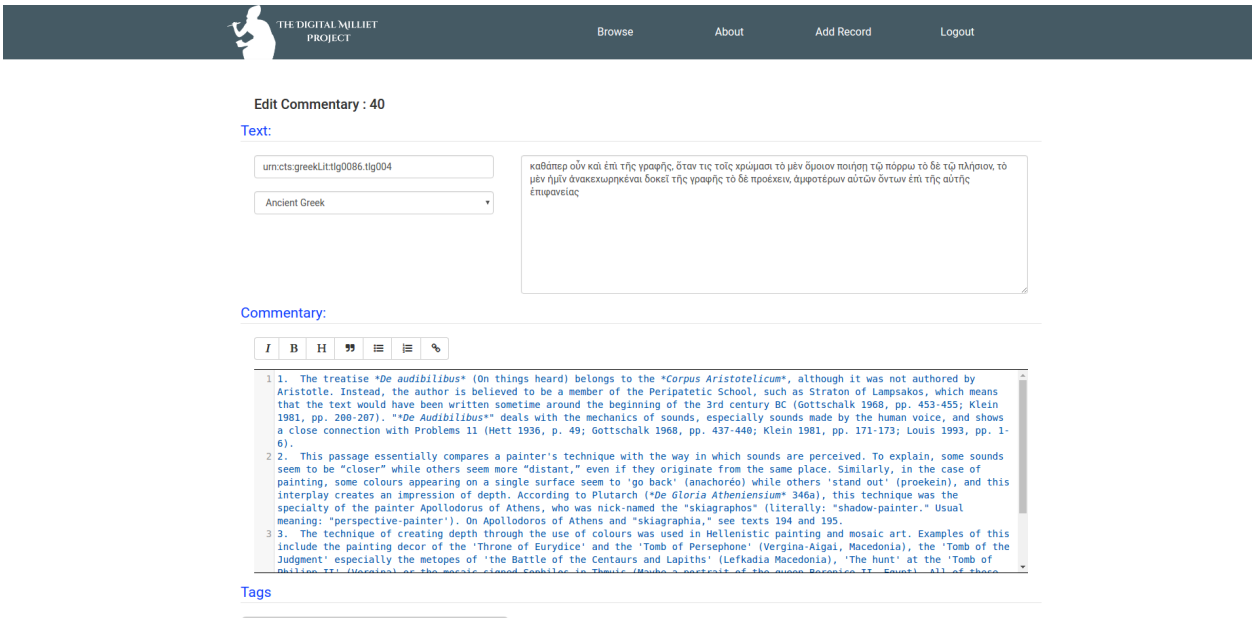
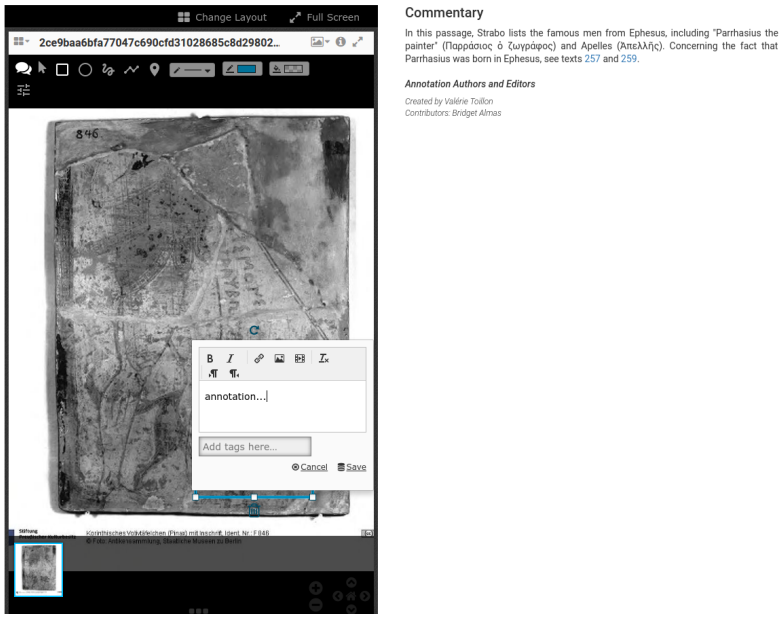


Image annotations can be viewed, added, edited and deleted directly using the Mirador viewer.



Click on the bubble icon to view annotations on the image. Hover your mouse over the marked up areas on the image to see the annotation text.

If you are logged in you can click Edit or Delete to edit or delete the image annotation.

You use the drawing tools in the Mirador viewer to create new annotations. Select a tool and drag the mouse to highlight the region of interest on the image. When you release the mouse the annotation dialog will popup and you can enter and save your annotation text.

CHAPTER 2

Database Schema

The Digital Milliet stores all data in MongoDB.

Digital Milliet commentary entries are stored in the *annotations* collection.

Author Records are stored in the collection named in the Digital Milliet config file setting *AUTHORS_COLLECTION*.

IIIF Image annotations are stored in the *mirador* collection.

(A future enhancement to externalize all collection names is requested in https://github.com/perseids-project/digital_milliet/issues/58)

The schema for the database objects is depicted here:

See also the test fixtures for examples of database entries.

CHAPTER 3

Modules

class digital_milliet.lib.commentaries.**CommentaryHandler**(*db=None*, *authors=None*, *config=None*, *auth=None*)

Parses data for retrieval/storage to/from the database

__init__(*db=None*, *authors=None*, *config=None*, *auth=None*)

CommentaryHandler object

Parameters

- **db** (*PyMongo*) – Mongo Db Handle
- **authors** (AuthorBuilder) – helper for building new Author records
- **config** (*dict*) – configuration dictionary

__weakref__

list of weak references to the object (if defined)

create_commentary(*form*)

Save a new set of annotations from the input form

Parameters **form** (*dict*) – key/value pairs from input form

Returns the Milliet number for the saved annotations or None if the record couldn't be saved

Return type string

create_tag_annotation(*tag*, *target*, *creator*, *date*)

Create a tag annotation

Parameters

- **tag** (*string*) – the tag (text or a URI)
- **target** (*string*) – the target of the annotation
- **creator** (*dict*) – the creator of the annotation
- **date** (*date*) – the date the annotation was created

Returns Annotation content to set at annotation["tags"]

form_to_OpenAnnotation(*form*)

Make a structure for the annotation from a set of key/value pairs

Parameters **form** (*dict*) – key/value pairs from the form

Returns the annotation

Return type dict

format_manifests_from_form(*manifest_uri*, *publisher*, *date*, *milnum*, *update_anno=None*)

Helper to format IIIF Manifests given a form

Parameters

- **manifest_uri** – Manifest URI
- **publisher** – Publisher
- **date** – Current date (Isocode)
- **milnum** – Current milnum

Returns Value to set at annotation["images"]

format_person_from_authentificated_user()

Make a Person for an annotation (i.e for contributor or creator) Uses the URI identifier for the user of the currently authenticated session

Returns Person properties suitable for inclusion in the annotation

Return type dict

format_translation_annotation(*num*, *milnum*, *text*, *uri*, *own_uri*, *lang*)

Build the body of a translation annotation.

Parameters

- **num** (*string*) – the translation identifier (t1 or t2)
- **milnum** (*string*) – the Milliet number for the annotation
- **text** (*String*) – the text of the translation (None if uri or own_uri is supplied)
- **uri** (*string*) – the uri of a translation - this is expected to be a CTS URN that appears in the linked cts repository
- **own_uri** (*string*) – an user-supplied uri for a translation - this is for an externally linked translation text
- **lang** (*string*) – the language code of the translation ('fra' or 'eng')

Returns the body of the translation annotation

Return type string (for a URI) or dict (if an embedded body)

format_uri(*milliet_id*, *subcollection_id=None*)

Make a Cite Collection URI for an annotation

N.B. this is not a valid implementation of the CITE protocol, as it does not support CITE collections. Future implementations should consider replacing this with a different identifier syntax.

Param milliet_id: The Milliet number

Type milliet_id: string

Param subcollection_id: the subcollection identifier (e.g. commentary, bibliography, etc.)

Type string

Returns the compiled URI

Return type string

generate_uuid()

Create a unique id for an annotation

Returns uid

Return type string

get_existing_tags()

List all existing tag body values

Returns tags and semantic tags

Return type tuple

get_milliet(*milliet_id*, *simplify=True*)

Get the first set of annotations that target the supplied Milliet Number

Parameters

- **milliet_id** – Milliet Number
- **simplify** (*bool*) – If set to True, simplify for the view

Returns Tuple where first element is the set of annotations and the second the author informations

Return type (dict, dict)

Raises **404 Not Found Exception** – if the annotation is not found

get_milliet_identifier_list()

List all known milliet numbers

Returns List of Milliet Numbers and their commentary ID ?

Return type tuple

get_surrounding_identifier(*cid*)

Given a Milliet number, return the previous and next numbers available

Parameters **cid** (*string*) – Milliet number

Returns pair of Milliet numbers

Return type (string, string)

remove_milliet(*milliet_id*)

Remove the annotation set that targets the supplied Milliet Number

Parameters **millnum** – Milliet Number

Returns the number of records removed

Return type int

Raises **404 Not Found Exception** – if the annotation is not found

retrieve_millietId_in_commentaries(*commentaries*)

Extract a sorted list of Milliet ID from a set of commentary annotations

Parameters **commentaries** (*list*) – set of commentary annotations

Returns sorted list of extracted Milliet numbers

Return type list

search (*query*, *tags=None*)
Search commentary record (Filters are exclusive) currently only searching in tags is supported

Parameters

- **query** – String to search
- **tags** – Search in tags

Returns List of matching records

simplify_milliet (*annotation_set*)
Parse a db record into a dict setup for views

Parameters **annotation_set** (*dict*) – the db record

Returns Parsed version of the record

Return type dict

update_commentary (*form*)
Save an edited set of annotations to the db

Parameters **form** (*dict*) – key/value pairs from edit form

Returns True if successful False if not

Return type bool

update_contributors (*annotation_dict=None*)
Update the contributors for an annotation

Inserts a Person object for the currently authenticated user if she doesn't already appear as either creator or contributor.

Parameters **annotation_dict** (*dict*) – the annotation to update

validate_annotation (*annotation*)
Validate the structure of an annotation.

This is not foolproof but it attempts to catch some errors that could come in from mistakes in data entry. It would be good to make sure these all couldn't occur to begin with.

Parameters **annotation** (*dict*) – the annotation record

Returns True if valid False if not

Return type bool

class digital_milliet.lib.author_builder.**AuthorBuilder** (*db=None*, *catalog=None*, *collection_name='annotation'*, *app=None*)
Provides methods for building new Author records in the database

__init__ (*db=None*, *catalog=None*, *collection_name='annotation'*, *app=None*)
Constructor

Parameters

- **db** (*PyMongo*) – Mongo Db Handle
- **catalog** (*Catalog*) – Catalog API Manager

__weakref__
list of weak references to the object (if defined)

author_db_build(*data_dict*)
Adds or Updates Author Records in the Annotation Database

Author Records contain authority name and work information and are populated as annotations referencing an author and work are added to the annotator store so that they can be used for browsing

Parameters **data_dict** (*dict*) – the full annotation

author_list()
Get a list of authors

Returns List of authors record

collection
Quick access to Mongo collection

get_author(*cts_id*)
Retrieve an author record by CTS ID

Parameters **cts_id** – CTS Identifier

Returns Author Record

get_author_by_mongoId(*_id*)
Retrieve an author record by Mongo Id

Parameters **_id** – Mongo Unique Identifier

Returns Author Record

make_author(*resp*)
" Make an Author db record from a catalog record and insert it in the database

Parameters **resp** (*dict*) – the response from teh catalog lookup

Returns the new Author db record

Return type dict

make_work(*work_id*, *millnum*, *pasg*)
Make a work record from a catalog record

Parameters

- **work_id** (*string*) – the CTS URN of a work
- **millnum** (*string*) – the Milliet number
- **pasg** (*string*) – the passage component from the work

Returns the work record

Return type dict

process_comm(*comm_list*)
Extract a sorted list of milliet numbers from a set of commentary annotations

Parameters **comm_list** (*list*) – set of commentary annotations

Returns sorted list of milliet numbers

Return type list

remove_milliet_id_from_author(*millnum*)
Remove milliet number mapping from an author record

Parameters **millnum** (*string*) – the milliet number to remove

Returns Number of mappings removed

search(*query*, *name=None*, *works=None*, *milliet_id=None*)
Search authors record (Filters are exclusive)

Parameters

- **query** – String to search
- **name** – Search in Name
- **works** – Search in Works

Returns List of matching records

update_author(*cts_id*, *author_record*)
Update author identified by CTS_ID

Parameters

- **cts_id** – CTS Identifier
- **author_record** – Updated Author Record

Returns Result of update

class digital_milliet.lib.catalog.**Catalog**(*app=None*)
Provides an interface to a Catalog API Endpoint which can lookup author and work records by CTS URN

__init__(*app=None*)
Constructor

Parameters **app** (*Flask*) – The Flask App

__weakref__
list of weak references to the object (if defined)

lookup_author(*urn=None*)
Looks up an Author by authority id in the remote Catalog API endpoint

Parameters **urn** (*string*) – The authority id (i.e textgroup CTS URN)

Returns response from the API (this should be abstracted)

Return type dict

lookup_work(*urn=None*)
Looks up an Work by authority id in the remote Catalog API endpoint

Parameters **urn** (*string*) – The authority id (i.e work CTS URN)

Returns response from the API (we should abstract this)

Return type dict

class digital_milliet.lib.oauth.**OAuthHelper**(*app*)
Helper class providing OAuth2 functionality to the application Implements flask_oauthlib.client

__init__(*app*)
Constructor

Parameters **app** (*Flask*) – the wrapped flask app

__weakref__
list of weak references to the object (if defined)

static current_user()
Gets the current user from the session

Returns { uri => <uri>, name => <name> }

Return type dict

static oauth_required(*f*)
decorator to add to a view to require an oauth user

Returns decorated function

Return type func

static oauth_token(*token=None*)
tokengetter function

Parameters **token** (*string*) – the Oauth token

Returns the current access token

Return type string

r_oauth_authorized()
Route for OAuth2 Authorization callback

Returns renders template

r_oauth_login()
Route for OAuth2 Login

Parameters **next** (*string*) – next url

Returns Redirects to OAuth Provider Login URL

static r_oauth_logout()
Route to clear the oauth data from the session

Parameters **next** (*string*) – next url

Returns redirects to next or renders template

user_in_community(*user_communities=None*)
Checks to see if the user is the authorized community for editing

This is a hack specific to the Perseids OAuth provider used as a way to limit editing of DM records to members of a specific community in Perseids Eventually editing could be delegated entirely to Perseids

Returns True if the user name is listed in the configured community members, False if the user name is not listed

Return type bool

class digital_milliet.lib.mirador.**Mirador**(*db*, *app*, *parser*)
Parses data for retrieval/storage to/from the database

__init__(*db*, *app*, *parser*)
Mirador object

Parameters

- **db** (*PyMongo*) – Mongo Db Handle
- **app** (*Flask*) – Flask App
- **parser** (CommentaryHandler) – CommentaryHandler

__weakref__
list of weak references to the object (if defined)

create()
Create View

Returns Recorded Data

delete()
Delete a record

Returns Status of deletion

static dump(*content*, *code=200*)
(View system) Returns a response in json with given code

Parameters

- **content** – BSON encodable object
- **code** – HTTP Status Code

Returns Response

from_collection(*digitial_milliet_id*)
Retrieve a list of annotations from a collection

Parameters **digitial_milliet_id** (*str*) – ID of the Digital Milliet Collection

Returns List of annotation

get(*image_uri=None*, *anno_id=None*, *_id=None*, *single=False*)
Retrieve annotations

Parameters

- **image_uri** (*str*) – URI of the canvas
- **anno_id** (*str*) – Public Identifier of the annotation
- **_id** (*str*) – Private Identifier of the annotation
- **single** (*bool*) – Retrieve a single annotation instead of a list

Returns List of Annotations matching the filters

search()
Search View

Returns Result of search

static simpleFormat(*oAnnotation*)
Simplify the format of the annotation (Removes unnecessary information for Mirador)

Parameters **oAnnotation** – Annotation to simplify

Returns Simpler Annotation

update()
Update an annotation

Returns Updated Record

CHAPTER 4

Indices and tables

- genindex
- modindex
- search

Python Module Index

d

Index

Symbols

A

C

D

F

G

L

M

O

P

R

S

U

V