Readux Documentation

Release 0.6.0

Emory University Libraries

Aug 01, 2017

Contents

Contents:

CHAPTER 1

Code Documentation

Annotation

An implementation of the Annotation store API for Annotator.js.

Models

class annotator_store.models.**AnnotationQuerySet**(*model=None*, *query=None*, *using=None*, *hints=None*)

Custom QuerySet for *Annotation*

visible_to(*user*)

Return annotations the specified user is allowed to view. Objects are found based on view_annotation permission and

annotations; users can access only their own annotations or those where permissions have been granted to a group they belong to.

Note: Due to the use of guardian.shortcuts.get_objects_for_user(), it is recommended to use this method first; it does combine the existing queryset query, but it does not chain as querysets normally do.

---

visible_to_group(*group*)

Return annotations the specified group is allowed to view. Objects are found based on view_annotation permission and per-object permissions.

Note: Due to the use of guardian.shortcuts.get_objects_for_user(), it is recommended to use this method first; it does combine the existing queryset query, but it does not chain as querysets normally do.

---

last_created_time()
Creation time of the most recently created annotation. If queryset is empty, returns None.

last_updated_time()
Update time of the most recently created annotation. If queryset is empty, returns None.

class annotator_store.models.**AnnotationManager**
Custom Manager for *Annotation*. Returns *AnnotationQuerySet* as default queryset, and exposes *visible_to()* for convenience.

visible_to(*user*)
Convenience access to *AnnotationQuerySet.visible_to()*

visible_to_group(*group*)
Convenience access to *AnnotationQuerySet.visible_to_group()*

class annotator_store.models.**BaseAnnotation**(**args*, ***kwargs*)
Django database model to store Annotator.js annotation data, based on the annotation format documentation.

UUID_REGEX = '[0-9a-f]{8}-[0-9a-f]{4}-[1-5][0-9a-f]{3}-[89ab][0-9a-f]{3}-[0-9a-f]{12}'
regex for recognizing valid UUID, for use in site urls

schema_version = 'v1.0'
annotation schema version: default v1.0

id
unique id for the annotation; uses uuid.uuid4()

created
datetime annotation was created; automatically set when added

updated
datetime annotation was last updated; automatically updated on save

text
content of the annotation

quote
the annotated text

uri
URI of the annotated document

user
user who owns the annotation when serialized, id of annotation owner OR an object with an 'id' property

extra_data
any additional data included in the annotation not parsed into specific model fields; this includes ranges, permissions, annotation data, etc

common_fields = ['text', 'quote', 'uri', 'user']
fields in the db model that are provided by annotation json when creating or updating an annotation

internal_fields = ['updated', 'created', 'id', 'user', 'annotator_schema_version']
internal fields that are not set from values provided by annotation json when creating or updating

get_absolute_url()
URL to view this annotation within the annotation API.

text_preview()
Short preview of annotation text content

uri_link()
URI as a clickable link

related_pages
convenience access to list of related pages in extra data

classmethod create_from_request(*request*)
Initialize a new *Annotation* based on data from a django.http.HttpRequest.

Expects request body content to be JSON; sets annotation user based on the request user.

update_from_request(*request*)
Update attributes from data in a django.http.HttpRequest. Expects request body content to be JSON. Currently does *not* modify user.

handle_extra_data(*data*, *request*)
Handle any "extra" data that is not part of the stock annotation data model. Use this method to customize the logic for creating and updating annotations from request data.

NOTE: request is passed in to support permissions handling when object-level permissions are enabled.

info()
Return a collections.OrderedDict of fields to be included in serialized JSON version of the current annotation.

class annotator_store.models.**AnnotationWithPermissions**(**args*, ***kwargs*)
Mix-in annotation class to provide object-level permissions handling via django-guardian

permission_to_codename = {'read': 'view_annotation', 'admin': 'admin_annotation', 'update': 'change_annotation
map annotator permissions to django annotation permission codenames

codename_to_permission = {'change_annotation': 'update', 'delete_annotation': 'delete', 'view_annotation': 'read'
lookup annotation permission mode by django permission codename

info()
Update default annotation info to include permissions

save(**args*, ***kwargs*)
Extend default save method to ensure annotation user has access to edit and update their own annotation.

handle_extra_data(*data*, *request*)
Handle any "extra" data that is not part of the stock annotation data model. Use this method to customize the logic for updating an annotation from request data.

user_permissions()
Queryset of guardian.model.UserObjectPermission objects associated with this annotation.

group_permissions()
Queryset of guardian.model.GroupObjectPermission objects associated with this annotation.

assign_permission(*permission*, *entity*)
Wrapper around guardian.shortcuts.assign_perm(). Gives the specified permission to the specified user or group on the current object.

db_permissions(*permissions*)
Convert annotation permission data into actionable django permissions using guardian per-object permissions.

permissions_dict()
Convert stored guardian per-object permissions into annotation permission dictionary format

user_has_perm(*user*, *permission*)
Check if a user has a specific permission on this object.

class annotator_store.models.**AnnotationGroup**(**args*, ***kwargs*)
Annotation Group; extends django.contrib.auth.models.Group.

Intended to facilitate group permissions on annotations.

notes
optional notes field

created
datetime annotation was created; automatically set when added

updated
datetime annotation was last updated; automatically updated on save

class annotator_store.models.**Annotation**(*id*, *created*, *updated*, *text*, *quote*, *uri*, *user*, *extra_data*)

Views

class annotator_store.views.**AnnotationIndex**(***kwargs*)
Annotator store API index view, with information and links for API urls.

class annotator_store.views.**Annotations**(***kwargs*)
API annotations view.

On GET, lists annotations. On AJAX POST with json data in request body, creates a new annotation.

Users must be logged in to create new annotations, and can only view their own annotations.

get(*request*)
List viewable annotations as JSON.

post(**args*, ***kwargs*)
Create a new annotation via AJAX.

class annotator_store.views.**AnnotationView**(***kwargs*)
Views for displaying, updating, and removing a single `Annotation`. All views require that the user be logged in and own the annotation being viewed, updated, or deleted.

get(*request*, *id*)
Display the JSON information for the requested annotation.

put(*request*, *id*)
Update the annotation via JSON data posted by AJAX.

delete(*request*, *id*)
Remove the annotation. On success, returns a 204 No Content response as per the annotator store API documentation.

class annotator_store.views.**AnnotationSearch**(***kwargs*)
Search annotations and display as JSON. Results are restricted to annotations the users has permission to view (currently only annotations owned by the user for everyone other than superusers).

The following search fields are currently supported:

- uri (exact match)
- text (case-insensitive partial match)
- quote (case-insensitive partial match)
- user (exact match on username)
- keyword: case-insensitive partial match on text, quote, or with extra data (e.g., to match tags)

Search results can be limited by specifying `limit` or `offset` parameters.

Utils

annotator_store.utils.**absolutize_url**(*local_url*)

Convert a local url to an absolute url, with scheme and server name, based on the current configured `Site`.

Parameters **local_url** – local url to be absolutized, e.g. something generated by `reverse()`

annotator_store.utils.**user_passes_test_ajax_403**(*test_func*, *login_url=None*, *redirect_field_name='next'*)

Variation on `django.contrib.auth.decorators.user_passes_test()`. If the user is already logged in but has insufficient privileges, returns a 403 Forbidden response because redirecting to log in is not useful. If request is ajax user is not authenticated, returns a 401 because redirecting an ajax request to login is also not useful.

Partially adapted from/inspired by `eulcommon` implementation. http://eulcommon.readthedocs.io/en/0.17.0/djangoextras.html#module-eulcommon.djangoextras.auth

annotator_store.utils.**permission_required**(*perm*, *login_url=None*)

Version of `django.contrib.auth.decorators.permission_required()` that uses `user_passes_test_with_ajax_403()` for better http response codes and ajax handling.

CHAPTER 2

CHANGELOG

Release 0.6

- Creating new annotations via annotator API now requires the user to have the *annotator_store.add_annotation* permission. (Previously any logged in user could create annotations.)
- Any annotation edits (create, update, delete) made via annotator API now generate Django admin log entries.

Release 0.5

Initial release of *annotator_store* as a stand-alone django application, refactored out of the Readux codebase.

- Support for custom Annotation model via **ANNOTATOR_ANNOTATION_MODEL** setting and *annotator_store.models.BaseAnnotation* abstract model
- Configurable support for normal Django permissions *or* per-item permissions via **django-guardian** (*NOTE* that per-item permissions functionality is not fully tested or supported and should be considered alpha quality)
- Annotation text and quote fields marked as optional for use in Django admin
- Supports both Python 3.x and Python 2.7
- Includes example templates with code for initializing an annotator.js instance and connecting it to annotator_store as a backend.

annotator_store is a Django application meant for use within a Django project as an annotator.js 2.x annotation store backend, and implements the Annotator Storage API.

annotator_store was originally develop as a component of Readux.

CHAPTER 3

License

This software is distributed under the Apache 2.0 License.

Installation

Use pip to install from GitHub:

```
pip install git+https://github.com/Princeton-CDH/django-annotator-store.git
↪#egg=annotator_store
```

Use branch or tag name, e.g. `@develop` or `@1.0`, to install a specific tagged release or branch:

```
pip install git+https://github.com/Princeton-CDH/django-annotator-store.git@develop
↪#egg=annotator_store
```

Configuration

Add *annotator_store* to installed applications and make sure that other required components are enabled:

```
INSTALLED_APPS = (
    ...
  'django.contrib.auth',
  'django.contrib.admin',
  'django.contrib.contenttypes',
  'django.contrib.sessions',
  'django.contrib.sites',
  'annotator_store',
    ...
)
```

Include the annotation storage API urls at the desired base url with the namespace:

```
from annotator_store import views as annotator_views

urlpatterns = [
    # annotations
    url(r'^annotations/api/', include('annotator_store.urls', namespace='annotation-
↪api')),
    # annotatorjs doesn't handle trailing slash in api prefix url
    url(r'^annotations/api', annotator_views.AnnotationIndex.as_view(), name=
↪'annotation-api-prefix'),
]
```

Run migrations to create annotation database tables:

```
python manage.py migrate
```

Note: If you want per-object permissions on individual annotations (rather than the standard django type-based permissions), you must also install *django-guardian* and include *guardian* in your **INSTALLED_APPS**. Per-object permissions must be turned on in Django settings by setting **ANNOTATION_OBJECT_PERMISSIONS** to True.

CHAPTER 4

Custom Annotation Model

This module is designed to allow the use of a custom Annotation model, in order to add functionality or relationships to other models within an application. To take advantage of this feature, you should extend the abstract model *annotator_store.models.BaseAnnotation* and configure your model in Django setings, e.g.:

```
ANNOTATOR_ANNOTATION_MODEL = 'myapp.LocalAnnotation'
```

If you want per-object permissions on your annotation model, you should extend *annotator_store.models.AnnotationWithPermissions* rather than the base annotation class.

Note: Per-object permissions require that a permissions plugin be included when you initialize your annotator.js Annotator object. That code is currently available as a plugin in the Readux codebase

Development instructions

This git repository uses git flow branching conventions.

Initial setup and installation:

- recommended: create and activate a python virtualenv:

```
virtualenv annotator-store
source annotator-store/bin/activate
```

- pip install the package with its python dependencies:

```
pip install -e .
```

CHAPTER 5

Unit Testing

Unit tests are run with py.test but use Django test classes for convenience and compatibility with django test suites. Running the tests requires a minimal settings file for Django required configurations.

- Copy sample test settings and add a **SECRET_KEY**:

```
cp ci/testsettings.py testsettings.py
```

- To run the tests, either use the configured setup.py test command:

```
python setup.py test
```

- Or install test requirements and use py.test directly:

```
pip install -e '.[test]'
py.test
```

CHAPTER 6

Sphinx Documentation

- To work with the sphinx documentation, install *sphinx* directly via pip or via:

```
pip install -e '.[docs]'
```

- Documentation can be built in the *docs* directory using:

```
make html
```

CHAPTER 7

Indices and tables

- genindex
- modindex
- search

Python Module Index

a

Index