Content Tagging

System-defined Taxonomies

Open edX development

Prepared for Axim by OpenCraft

30 May 2023

Context

The System-defined are closed taxonomies created by the system. This taxonomies follows the Taxonomy Data Architecture and have the following characteristics:

- Taxonomy administrators cannot add, delete or edit tags or taxonomy metadata when it is system-defined.
- Hardcoded or auto-generated from the codebase or core data models.
- Some may be visible and some may not on the UI.
- When tagging a content, it must be filled automatically by the platform and some must not be editable by the users.

System-defined Taxonomies

Each system-defined taxonomy has characteristics depending on its use and its context:

- **Tags creation**:
  - **Hardcoded by fixtures/migrations:** Its creation is completely static and depends on migrations. If we need to delete, edit or add tags, it is also done through migrations. It's possible to use get_tags to configure which tags are going to be shown to the user. E.g Language
  - **Free-form tags:** This taxonomy depends on a core data model, the tags are an unique ID of the model, but a Tag is not created per model, but as many tags as possible are allowed. This reduces complexity when many objects are created or are highly changeable. E.g. Author
- **Visible:** When adding tags to content, True if the content author can see these taxonomy tags; False if the content author can not see these taxonomy tags.
- **Automatic tagging:** The platform tags automatically a content from its metadata.
- **Allow multiple:** Indicates if the taxonomy allows multiple tags per content item.
- **Editable in the content:** True if the content author can edit this field on the content: False if the content author can not edit this field on the content.

The following are the system-defined taxonomies for the Open edX platform. There are some that are examples on the MVP requirements document

Language

- **Tags creation:** Hardcoded by fixtures/migrations

- **Visible:** True
- **Automatic tagging:** From the content author preferred language or browser language
- **Editable in the content:** True
- **Allow multiple:** False
- **Tags:**
    - Create all possible locales (ISO 639-1) and in get_tags() only return the enabled languages from Django LANGUAGES

~~Content type~~

- ~~**Tags creation:** Hardcoded by fixtures/migrations~~
- ~~**Visible:** False~~
- ~~**Automatic tagging:** From the metadata of the content~~
- ~~**Editable in the content:** False~~
- ~~**Allow multiple:** True~~
- ~~**Tags (Ref):**~~
    - ~~Text~~
    - ~~Video~~
    - ~~Discussion~~
    - ~~Problem~~
    - ~~Library~~

Organization

- **Tags creation:** Free-form tags
- **Visible:** False
- **Automatic tagging:** From the content author organization
- **Editable in the content:** False
- **Allow multiple:** False
- **Tags:** The IDs of the organizations with this structure: org:<ID>

Author

- **Tags creation:** Free-form tags
- **Visible:** False
- **Automatic tagging:** From the content author user id
- **Editable in the content:** False
- **Allow multiple:** False
- **Tags:** The IDs of the content author users with this structure: *user:<ID>*

Data Structures

SystemTaxonomy

This subclasses Taxonomy and adds some fields and properties to meet the different characteristics of the system-defined taxonomies. It's used to create system-taxonomies in different cases.

Fields

- visible (bool, default False)
- can_changed_on_content (bool, default False)
- allow_multiple (from Taxonomy)
- allow_free_text (from Taxonomy)

Properties

is_created_by_migrations: boolean

If allow_free_text == False

is_free_form: boolean

If allow_free_text == True

Methods

get_tags(self): List[Tag]

Returns a list of available tags. This function is implemented in Taxonomy, but can be overwritten to handle special cases.

validate_object_tag(ObjectTag: object_tag): boolean

Function used in tag_object and get_object_tags to make special validations within these taxonomies. This function is implemented in Taxonomy, but can be overwritten to handle special cases.

LanguageSystemTaxonomy

Subclasses SystemTaxonomy.

Methods

get_tags(self): List[Tag]

Overwrites Taxonomy get_tags. Returns a list of tags of the available languages for the platform. It can be built from Django LANGUAGES.

validate_object_tag(ObjectTag: object_tag): boolean

Makes a validation with Django LANGUAGES

OrganizationSystemTaxonomy

Subclasses SystemTaxonomy.

Methods

validate_object_tag(ObjectTag: object_tag): boolean

Extract the ID of each tag and verify that the organization exists.

ContentTypeSystemTaxonomy

Subclasses SystemTaxonomy.

Methods

validate_object_tag(ObjectTag: object_tag): boolean

Verify if the tags are on the list of values.

AuthorSystemTaxonomy

Subclasses SystemTaxonomy.

Methods

validate_object_tag(ObjectTag: object_tag): boolean

Extract the ID of each tag and verify that the User exists.

ContentSystemTaxonomyMixin

This class is used to create the Content-side models.

Methods

extract_tag(self, taxonomy, content): Tag

Returns the tag that should be saved according to the metadata of the content.

Content-side models

Each content that is tagged will have a class inherited from ContentSystemTaxonomyMixin. In order to manage the functions of automatic tagging and other functions that each case requires. Also we need to inherit from PipilineStep if the taxonomy allows auto tagging. You can find all about the auto tagging in a later section. This classes lives under `openedx.features.tagging.` Currently these classes are used only for auto tagging. For example:

- For Course block we have*:*
    - CourseLanguageSystemTaxonomy(ContentSystemTaxonomyMixin)
    - CourseOrganizationSystemTaxonomy(ContentSystemTaxonomyMixin)
    - CourseContentTypeSystemTaxonomy(ContentSystemTaxonomyMixin)
    - CourseAuthorSystemTaxonomy(ContentSystemTaxonomyMixin)

How to create a System-defined taxonomy

You must create a content-side model for each content and taxonomy to be used. All system-defined taxonomies must be created initially in a fixture from the Content-side models.

If you need to edit the characteristics or the metadata of the taxonomy you need to add the changes on a migration.

Tags creation

Hardcoded by fixtures/migrations

- If the tags don't change over the time, you can create all on a fixture (e.g Languages).

- If the tags change over the time, you can create all on a migration. If you edit, delete, or add new tags, you should also do it in a migration.

Free-form tags

This taxonomy depends on a core data model, but simplifies the creation of Tags by allowing free-form tags but we can validate the tags using the validate_tags method. Here there is no need to do configuration since any value that comes from a field of the associated model is allowed.

For example we can put the *Author* taxonomy associated with the *User* model and use the *ID* field as tags. Also we can validate if an *User* if a user still exists or has been deleted over time.

How to configure the “Automatic tagging” feature

Auto-tagging occurs when tags are added to content automatically using the content's metadata. We can have the following use cases:

Hidden Taxonomy

When *visible* is *False*. In this case, since the taxonomy is not visible in the content, the content author cannot add or change a tag, so we can auto tag the content after saving it. For this we will use Open edX filters to add a tag after a content creation/edition. So we need to add some filters for the content creation and edition.

Later, we can implement the run_filter method and follow this doc to configure the filter

```
def run_filter(self, content):
    value = self.extract_tag(content)
    return tag_object(self, [[value]], content.id)
```

Visible Taxonomy

When *visible* is *True*. In this case, as it is visible to the author of the content, it is necessary to fill in this tag at the time of rendering the field. In this example we will use CourseLanguageSystemTaxonomy.

```
course_lang_taxonomy = …  # CourseLanguageSystemTaxonomy
creation = …. #  True if the content is being created.
if creation:
    selected = course_lang_taxonomy.extract_tag(course)
else:
    selected = get_object_tags(course_lang_taxonomy, unit.id,
valid_only=False),
render_input_field(
    label=course_lang_taxonomy.name,
    options=get_tags(course_lang_taxonomy),
    selected=selected,
    free_text=course_lang_taxonomy.allow_free_text,
    multiple=course_lang_taxonomy.allow_multiple,
)
```

Decisions

Architectural decisions are listed for discussion, and will be included in the ADR.

System-defined Taxonomy & Tags creation

It is necessary to define how to create and validate the System-defined taxonomies and their tags.

Decisions

Create a Content-side class that lives on `openedx.features.tagging`. You must create a content-side model for each content and taxonomy to be used. You can use ContentSystemTaxonomyMixin and the mixin of the Taxonomy (e.g.

LanguageSystemTaxonomy) to create the new class. Then, all system-defined taxonomies must be created initially in a fixture.

Each Taxonomy mixin has get_tags; to configure the valid tags, and validate_tags; to check if a list of tags are valid.

We have two ways to handle the tags in this type of taxonomies:

Hardcoded by fixtures/migrations

- If the tags don't change over the time, you can create all on a fixture (e.g Languages).
- If the tags change over the time, you can create all on a migration. If you edit, delete, or add new tags, you should also do it in a migration.

Free-form tags

This taxonomy depends on a core data model, but simplifies the creation of Tags by allowing free-form tags but we can validate the tags using the validate_tags method. For example we can put the *Author* taxonomy associated with the *User* model and use the *ID* field as tags. Also we can validate if an *User* still exists or has been deleted over time.

Reasons

- There are taxonomies that are fully static but large, so we can edit the tags that we can show to users.
- If the Taxonomy depends on a core data model (e.g Organizations), with free-form tags we don’t need to create every Tag. Also we can validate the content tags.

Rejected Options

Auto-generated from the codebase

Taxonomies that depend on a core data model and it is necessary to create a Tag for each object created.

System-defined automatic tagging

It is necessary to implement the automatic tagging functionality for the system-defined taxonomies when the associated content is created or edited.

Decisions

Use openedx-filters to call the auto tagging function after content creation/edition.

Reasons

- It's the best standard to use on the platform.
- It is simple and understandable.

Rejected Options

Use Django Signals

Implement a function to add the tag from the content metadata and register that function as a Django signal. Use openedx-filters is better in the edx context, but if there is other no-edX project that need to use openedx-tagging, can use the Django Signals approach:

```
def add_tag(self, content):
    value = content.id
    tag_object(self, [[value]], content.id)

def automatic_tag_content_handler(sender, instance, created,
taxonomy,**kwargs):
    taxonomy.add_tag(
        instance # Content
    )

def automatic_tag_content_signal_registration(model_class, taxonomy):
    post_save.connect(automatic_tag_content_handler,
sender=model_class, taxonomy=taxonomy)
```

And then we need to register a signal for each content that we will tag. In the following example we are registering signals for the *Organizations* taxonomy with the *Unit* and *Component* as contents:

```
course_org_taxonomy = …  # The CourseOrganizationSystemTaxonomy
library_org_taxonomy = … # The LibraryOrganizationSystemTaxonomy
automatic_tag_content_signal_registration(Course, org_taxonomy)
automatic_tag_content_signal_registration(Library,
library_org_taxonomy)
```

```
. . . # More contents here
```