Wagtail Approval Documentation

Release 0.2.11

Taylor C. Richberger

Oct 05, 2017

Contents:

CHAPTER 1

API

wagtailapproval.models

class **`ApprovalPipeline`**(**args*, ***kwargs*)

Bases: `wagtail.wagtailcore.models.Page`

This page type is a very simple page that is only used to hold steps

class **`ApprovalStep`**(**args*, ***kwargs*)

Bases: `wagtail.wagtailcore.models.Page`

Holds posts and facilitates the automatic moving to other steps in the same pipeline on approval and rejection.

`approve`(*obj*)

Run approval on an object

`automatic_approval`(*obj*)

Possibly runs processing on an object for automatic approval or rejection

`base_form_class`

alias of `StepForm`

`clean`()

Makes sure parents are the same

`fix_permissions`()

Set proper restrictions for the owned collection and all owned pages. Does not perform a save, so it can be safely used in a post_save signal.

`get_items`(*user*)

Gets an iterator of approval items, for rendering in templates. In practice, this returns a generator. If you need a stable view, use this to construct a list or tuple.

`reject`(*obj*)

Run rejection on an object

`release_ownership`(*obj*)

Release ownership of an object. This is idempotent.

`set_collection_group_privacy`(*private*)
Sets/unsets the collection group privacy

`set_page_group_privacy`(*page*, *private*)
Sets/unsets the page group privacy

`take_ownership`(*obj*)
Take ownership of an object. Should run all relevant processing on changing visibility and other such things. This is idempotent.

`transfer_ownership`(*obj*, *step*)
Give ownership to another step

class **`ApprovalTicket`**(**args*, ***kwargs*)
Bases: `django.db.models.base.Model`

A special junction table to reference an arbitrary item by uuid.

This is used to create an arbitrary approval/rejection URL, as it would be very difficult to do otherwise (as an approval step can own arbitrary pages and collection members with conflicting PKs otherwise). UUID is done for a minor security gain (prevent people from being able to try to act on arbitrary PKs, though that will be prevented through user privileges anyway, and the UUID should only be used for approvals and rejections, not GETs), as well as making the URL more opaque.

wagtailapproval.approvalitem

class **`ApprovalItem`**(*title*, *view_url*, *edit_url*, *delete_url*, *obj*, *step*, *typename*, *uuid*)
An Approval menu item, used for building the munu list, including links and all. Objects of this type should be added through the *`build_approval_item_list`* signal.

Parameters

- **`title`** (*`str`*) – The title as displayed in the list
- **`view_url`** (*`str`*) – The URL to view the item.
- **`edit_url`** (*`str`*) – The URL to edit the item.
- **`delete_url`** (*`str`*) – The URL to delete the item.
- **`obj`** – The item itself.
- **`step`** (`ApprovalStep`) – The step for this item.
- **`typename`** (*`str`*) – The type name of the item.
- **`uuid`** (*`uuid.UUID`*) – The UUID for this item, the pk for *`ApprovalTicket`*

`delete_url`

`edit_url`

`obj`

`step`

`title`

`typename`

`uuid`

`view_url`

`get_user_approval_items`(*user*)

Get an iterable of all items pending for a user's approval.

Parameters **`user`** (`User`) – A user object whose groups are to be checked for appropriate steps

Return type Iterable[*ApprovalItem*]

Returns All the items that this user can approve or reject.

wagtailapproval.apps

class `WagtailApprovalConfig`(*app_name*, *app_module*)

Bases: `django.apps.config.AppConfig`

Simply imports signals

`name` **= 'wagtailapproval'**

`ready`()

wagtailapproval.forms

class `StepForm`(**args*, ***kwargs*)

Bases: `wagtail.wagtailadmin.forms.WagtailAdminPageForm`

This is used to filter the approval and rejection steps so that only siblings may show. Proper validation on saving is performed in the Step model itself.

`base_fields` **= OrderedDict()**

`declared_fields` **= OrderedDict()**

`media`

wagtailapproval.menu

class `ApprovalMenuItem`(*label=u'Approval'*, *url=u'/admin/approval/'*, *classnames='icon icon-tick-inverse'*, *order=200*, ***kwargs*)

Bases: `wagtail.wagtailadmin.menu.MenuItem`

The menu item that shows in the wagtail sidebar

`is_shown`(*request*)

Only show the menu if the user is in an owned approval group

`media`

wagtailapproval.signals

This is all the signals supported by wagtailapproval. Examples of most of the important ones can be found directly in *`wagtailapproval._signals`*.

Signals are documented here as the functions that catch them. `sender`, `signal`, and `**kwargs` are ommitted for brevity. Some signals expect return values and may misbehave if the signal handlers don't return what they are expected to.

step_published(*instance*)
Sent when a step is published.

Parameters **instance** (ApprovalStep) – The instance that was published

pipeline_published(*instance*)
Sent when a pipeline is published.

Parameters **instance** (ApprovalPipeline) – The instance that was published

build_approval_item_list(*approval_step*, *user*)
Used when building the approval items. Should return an iterable of ApprovalItem instances. You may return any iterable, meaning generators are also acceptable.

Parameters

- **approval_step** (ApprovalStep) – The step to grab items from
- **user** (*User*) – The user to grab items for

Return type Iterable[*ApprovalItem*]

Returns An iterable of *ApprovalItem* instances

remove_approval_items(*approval_items*, *user*)
Can be used to implement custom filtering. Should return an iterable of ApprovalItem instances that the user doesn't want shown. The instances don't have to exactly match, but may support equality. It is preferable that you use the same object, though, as an `is` comparison is faster.

Parameters

- **approval_items** (*tuple[*ApprovalItem*]*) – The full list of approval items
- **user** (*User*) – The user to filter items for

Return type Iterable[*ApprovalItem*]

Returns An iterable of *ApprovalItem* instances

set_collection_edit(*approval_step*, *edit*)
Can be used to customize collection editing permissions. Use the edit kwarg, not the step's can_edit field, because they might not match.

Parameters

- **approval_step** (ApprovalStep) – The step to modify collection permissions for
- **edit** (*bool*) – whether editing is to be enabled or disabled

take_ownership(*approval_step*, *object*, *pipeline*)
Used for taking ownership by specific type. Do not work with ApprovalTicket here, as it's done automatically after this signal is called. This is done for permissions management.

Parameters

- **approval_step** (ApprovalStep) – The step to give the object to.
- **object** – The object to give to the step
- **pipeline** (ApprovalPipeline) – The pipeline for the ApprovalStep

release_ownership(*approval_step*, *object*, *pipeline*)
Used for releasing ownership by specific type. Do not work with ApprovalTicket here, as it's done automatically after this signal is called. This is used for permissions management.

Parameters

- **`approval_step`** (`ApprovalStep`) – The step to give the object to.
- **`object`** – The object to give to the step
- **`pipeline`** (`ApprovalPipeline`) – The pipeline for the ApprovalStep

`pre_transfer_ownership`(*giving_step*, *taking_step*, *object*, *pipeline*)
Sent before transferring ownership. This is done after *`pre_approve()`* or *`pre_reject()`*. This can be used for validation.

Parameters

- **`giving_step`** (`ApprovalStep`) – The step who will be releasing the object
- **`taking_step`** (`ApprovalStep`) – The step who will be taking the object
- **`object`** – The object to be transferred.
- **`pipeline`** (`ApprovalPipeline`) – The pipeline for the steps

`post_transfer_ownership`(*giving_step*, *taking_step*, *object*, *pipeline*)
Sent after transferring ownership. This is done before *`post_approve()`* or *`post_reject()`*. This should be used if you want to do something after each transfer.

Parameters

- **`giving_step`** (`ApprovalStep`) – The step that has released the object
- **`taking_step`** (`ApprovalStep`) – The step that has taken the object
- **`object`** – The object that has been transferred
- **`pipeline`** (`ApprovalPipeline`) – The pipeline for the steps

`pre_approve`(*giving_step*, *taking_step*, *object*, *pipeline*)
Sent before approval. This is done before *`pre_transfer_ownership()`*. This can be used for validation. If *`approve`* is run on an object that has no approval step, this will not be executed.

Parameters

- **`giving_step`** (`ApprovalStep`) – The step who will be releasing the object
- **`taking_step`** (`ApprovalStep`) – The step who will be taking the object
- **`object`** – The object to be transferred.
- **`pipeline`** (`ApprovalPipeline`) – The pipeline for the steps

`post_approve`(*giving_step*, *taking_step*, *object*, *pipeline*)
Sent after approval. This is done after *`post_transfer_ownership()`*. This should be used if you want to do something after each transfer (such as if `taking_step` is a step that is meant to perform some sort of automatic validation or automatic approval/rejection). If *`approve`* is run on an object that has no approval step, this will not be executed.

Parameters

- **`giving_step`** (`ApprovalStep`) – The step that has released the object
- **`taking_step`** (`ApprovalStep`) – The step that has taken the object
- **`object`** – The object that has been transferred
- **`pipeline`** (`ApprovalPipeline`) – The pipeline for the steps

`pre_reject`(*giving_step*, *taking_step*, *object*, *pipeline*)
Sent before rejection. This is done before *`pre_transfer_ownership()`*. This can be used for validation. If *`approve`* is run on an object that has no rejection step, this will not be executed.

Parameters

- **giving_step** (ApprovalStep) – The step who will be releasing the object
- **taking_step** (ApprovalStep) – The step who will be taking the object
- **object** – The object to be transferred.
- **pipeline** (ApprovalPipeline) – The pipeline for the steps

post_reject (*giving_step*, *taking_step*, *object*, *pipeline*)

Sent after rejection. This is done after *post_transfer_ownership()*. This should be used if you want to do something after each transfer (such as if `taking_step` is a step that is meant to perform some sort of automatic validation or automatic approval/rejection). If *approve* is run on an object that has no rejection step, this will not be executed.

Parameters

- **giving_step** (ApprovalStep) – The step that has released the object
- **taking_step** (ApprovalStep) – The step that has taken the object
- **object** – The object that has been transferred
- **pipeline** (ApprovalPipeline) – The pipeline for the steps

wagtailapproval.views

approve (*request*, *pk*)

check_permissions (*function*)

index (*request*)

Get all pending approvals that are relevant for the current user

reject (*request*, *pk*)

wagtailapproval.wagtail_hooks

register_admin_menu_item ()

register_admin_urls ()

take_ownership_if_necessary (*request*, *page*)

Checks the request user and takes ownership of the page if it is created by an owned user

wagtailapproval._signals

These are the signals used for internal behavior. Some of these are great places to start if you're looking at adding your own types to your pipeline.

add_document (*sender*, *approval_step*, ***kwargs*)

Builds the approval item list for documents

add_images (*sender*, *approval_step*, ***kwargs*)

Builds the approval item list for images

`add_pages`(*sender*, *approval_step*, ***kwargs*)
: Builds the approval item list for pages

`approvalticket_cascade_delete`(*sender*, *instance*, ***kwargs*)
: This deletes objects from `ApprovalTicket` if they are deleted, to avoid leaking space (a deleted object would otherwise never be freed from the ticket database, as cascades don't work for `GenericForeignKey` without a `GenericRelation`). Essentially, this is a custom cascade delete.

`assert_page_live`(*sender*, *giving_step*, *taking_step*, *object*, *pipeline*, ***kwargs*)

`catch_collection_objects`(*sender*, *instance*, *created*, ***kwargs*)
: If newly-created objects are created inside of a collection that is owned by an ApprovalStep, it will automatically take ownership of those objects

`delete_owned_group_and_collection`(*sender*, *instance*, ***kwargs*)
: This deletes the owned group and collection from `ApprovalStep` when the step is deleted.

`delete_owned_user`(*sender*, *instance*, ***kwargs*)
: This deletes the owned user from `ApprovalPipeline` when the pipeline is deleted.

`fix_restrictions`(*sender*, *instance*, ***kwargs*)
: Update ApprovalStep restrictions on a save.

`release_page_permissions`(*sender*, *approval_step*, *object*, *pipeline*, ***kwargs*)

`send_published_signals`(*sender*, *instance*, ***kwargs*)
: This simply watches for a published step or pipeline, and sends a `pipeline_published()` or `step_published()` signal for it.

`set_document_collection_edit`(*sender*, *approval_step*, *edit*, ***kwargs*)
: Sets collection permissions for documents

`set_image_collection_edit`(*sender*, *approval_step*, *edit*, ***kwargs*)
: Sets collection permissions for images

`setup_group_and_collection`(*sender*, *instance*, ***kwargs*)
: Create or rename the step's owned groups and collections

`setup_pipeline_user`(*sender*, *instance*, ***kwargs*)
: Setup an ApprovalPipeline user

`update_collection_ownership`(*sender*, *approval_step*, *object*, *pipeline*, ***kwargs*)
: Individual take_ownerships for each type should be implemented that also take the collection member. This is a fallback in case something doesn't work the way it should

`update_document_ownership`(*sender*, *approval_step*, *object*, *pipeline*, ***kwargs*)

`update_image_ownership`(*sender*, *approval_step*, *object*, *pipeline*, ***kwargs*)

`update_page_ownership`(*sender*, *approval_step*, *object*, *pipeline*, ***kwargs*)

CHAPTER 2

Wagtail Approval

This is a wagtail plugin for approval pipelines.

CHAPTER 3

What is this?

Essentially, this is a plugin for defining and enforcing flows of approval and editing. You can set up arbitrary “steps” in the flow, each step owning a group (and all the users which belong to the group). When a user creates a relevant object, their step will catch it and take ownership of it. When a user’s step owns an object, that user can then approve or reject objects as relevant. Steps can be made to make all of their owned objects private, so that items can be kept in an unpublished state until they are fully edited and approved (note that “Published” takes a different meaning here than Wagtail’s own). What this does as far as pages are concerned could be replicated by creating a “creation” page that is private and allowing approval and editing users to move pages into specific areas, but that is messy and prone to failure. This takes that process, avoids all the moving pieces, and puts the whole thing on rails.

CHAPTER 4

Does it work out of the box?

Wagtail Approval works out of the box for base wagtail (more specifically, it works for Images, Documents, and Pages). It can be extended to support any other collectable type you wish, as long as that type properly implements permissions for their collections (ie, respects the `add`, `change`, and `delete` permissions and also properly implements view restrictions)

CHAPTER 5

What does not work?

Wagtail Images are not made private when they are in their collection. This is an issue in Wagtail, and comes about because wagtail does not actually serve images. Images are instead served directly out of the Django media path.

CHAPTER 6

How do I get started?

You can get started with the following steps:

1. Create an `ApprovalPipeline` page.
2. Create a set of `ApprovalStep` pages inside the pipeline.
3. Link the steps together (after they are created and published) by their approval and rejection fields.
4. Create users and assign them to the groups created by the steps.
5. Give the groups that should have creation permissions the relevant perms for their types and pages that they should be able to create in.
6. Publish an object as a content creation user.

There should be no subclassing necessary. Appropriate extension should be doable entirely through signals. If you can't extend this in the way you need to through signals, it's probably a bug in this plugin.

CHAPTER 7

License

This project is licensed under the 2-clause BSD license, copyrighted by Absolute Performance, Inc. See the LICENSE document for more information.

Portions of code are copied from the wagtailnews project, and thus the wagtailnews attribution requirements are carried as well by this project:

CHAPTER 8

Indices and tables

- genindex
- modindex
- search

Python Module Index

Index