Django Public Admin

Release 0.0.4

Oct 08, 2020

Table of contents:

A public and read-only version of the Django Admin. A drop-in replacement for Django's native `AdminSite` and `ModelAdmin` for publicly accessible data.

CHAPTER 1

How does it work

- *public_admin.sites.PublicApp* wraps Django apps and models you want to make public.
- *public_admin.sites.PublicAdminSite* works as a clone of Django's native *AdminSite*, but it looks at the HTTP request and the URL to decide whether they should exist in a public and read-only dashboard.
- *public_admin.admin.PublicModelAdmin* work as a clone of Django's native *ModelAdmin*, but what it does is to stop actions that would create, edit or delete objects.

CHAPTER 2

Install

```
pip install django-public-admin
```

2.1 Usage

2.1.1 Declare which apps and models you want to make public

Let's say you have a Django app called `my_open_house` with models `Beverage` and `Snack` that you want their data to de public. Use *public_admin.sites.PublicApp* to declare that:

```
from public_admin.sites import PublicApp

public_app = PublicApp("my_open_house", models=("Beverage", "Snack"))
```

2.1.2 Create your *Django Public Admin* instance

Just like one would create a regular `admin.py`, you can create a module *public_admin.sites.PublicAdminSite* and *public_admin.admin.PublicModelAdmin*:

```
from public_admin.sites import PublicAdminSite, PublicApp


public_app = PublicApp("my_open_house", models=("beverage", "snack"))
public_admin = PublicAdminSite("dashboard", public_app)
```

The first argument is the name of this site in Django, and the second argument can be a single instance of *public_admin.sites.PublicApp* or a sequence of them.

2.1.3 Create and register your `PublicModelAdmin`

```
from public_admin.admin import PublicModelAdmin

from my_open_house.models import Beverage, Snack


class BeverageModelAdmin(PublicModelAdmin):
    # ...


class SnackModelAdmin(PublicModelAdmin):
    # ...


public_admin.register(Beverage, BeverageModelAdmin)
public_admin.register(Snack, SnackModelAdmin)
```

2.1.4 Add your *Django Public Admin* URLs

In your `urls.py`, import the *public_admin* (or whatever you've named it earlier) in your URLs file and create the endpoints:

```
from django.urls import path

from my_website.my_open_house.admin import public_admin


urlpatterns = [
    # ...
    path("dashboard/", public_admin.urls)
]
```

2.1.5 Templates

Django Public Admin comes with a template that hides from the UI elements related to non-logged-in users (elements such as login and logout links, recent actions panel, etc.). These templates are designed in a way to preserve the behavior of a regular instance of Django's native admin for logged-in users. To use it, add `"public_admin"` to your `INSTALLED_APPS` **before** `django.contrib.admin`:

```
INSTALLED_APPS = [
    "public_admin",
    "django.contrib.admin",
    # ...
]
```

If you decide not to use this template, you have to create your own `templates/admin/base.html` file to avoid errors when rendering the template. Django will fail, for example, in rendering URLs that do not exist, which would be the case for login and logout.

2.2 Example

There is an example app in Django Public Admin repository, inside the `example/` directory. This example is meant to be a straightforward use case, having *Django's native admin* running in parallel with *Django Public Admin*.

2.2.1 Requirements

- Git
- Python 3.6 or newer with Poetry (or other PEP 517 *pyproject.toml* compatible tool)

2.2.2 Running the example

First, clone the repository and install the dependencies:

```
git clone https://github.com/cuducos/django-public-admin.git
poetry install
```

Then start the application:

```
poetry run python example/manage.py runexample
```

The `runexample` command is a wrapper around Django's native `runserver`. It creates a temporary SQLite database, run migrations, creates a superuser, and collects static files *automagically* before spinning up the development server. If you are having trouble with this command, you can try to delete all these temporary files with `poetry run python manage.py cleanexample`.

Once the application is up and running, you can:

- Access the *Django's native admin*, **password protected** (username is `admin` and password is also `admin`) at localhost:8000/admin
- Access the *Django Public Admin*, with **no login needed** at localhost:8000/dashboard

You can add and edit data at `admin/`, while non-logged-in users can browse data at `dashboard/` with all the filters and perks of a Django Admin instance!

2.3 API

class `public_admin.sites.`**`PublicApp`**(*name*, *models*)

Holds the permission strings for each model in a Django app. *name* should be the name of a Django app as string, and *models* should be a sequence of strings with the name of the models to allowed in a public admin.

class `public_admin.sites.`**`DummyUser`**(*public_apps*, **args*, ***kwargs*)

Mimics the Django's native *AnonymousUser* injecting permissions to view objects from certain Django apps and models.'pubic_apps' should be a sequence of instances of *public_admin.sites.PublicApp*.

`has_module_perms`(*app_label*)

Only grant permission if the *app* was passed as a *public_admin.sites.PublicApp*.

`has_perm`(*permission*, *obj=None*)

Only grant permission if the app and model were passed in a *public_admin.sites.PublicApp*.

class `public_admin.sites.`**`PublicAdminSite`**(*name='public_admin'*, *public_apps=()*)

Mimics the Django's native *AdminSite* but removing URLs and permissions that does not match the idea of a public admin. *name* is the name of this admin site (the string Django uses to build the URL names, for example), and *pubic_apps* can be one instance of *public_admin.sites.PublicApp* or a sequence of them.

`admin_view`(*view*, *cacheable=False*)

Injects the *public_admin.sites.DummyUser* in every request in this admin site.

`has_permission`(*request*)

Blocks all non-GET requests.

`urls`

List the URLs in this admin site.

static **`valid_url`**(*url*)

This method removes URLs based on their path.

class `public_admin.admin.`**`PublicModelAdmin`**(*model*, *admin_site*)

This mimics the Django's native ModelAdmin but filters URLs that should not exist in a public admin, and deals with request-based permissions.

`get_urls`()

Filter out the URLs that should not exist in a public admin.

`has_add_permission`(*request*)

Denies permission to any request trying to add new objects.

`has_change_permission`(*request*, *obj=None*)

Denies permission to any request trying to change objects.

`has_delete_permission`(*request*, *obj=None*)

Denies permission to any request trying to delete objects.

`has_view_permission`(*request*, *obj=None*)

Only allows view requests if the method is GET

CHAPTER 3

References

- genindex
- modindex
- search

Index