
django-gdpr-assist Documentation

Release 1.0.1

Wildfish

Oct 23, 2018

Contents:

1	Installation	3
1.1	Django settings	3
2	Usage	5
2.1	Configure your models	5
3	The <code>PrivacyMeta</code> object	7
3.1	Registering automatically	7
3.2	Registering manually	7
3.3	Attributes	8
4	Anonymising objects	11
4.1	Model changes	11
4.2	How anonymisation works	11
4.3	Anonymising related objects	12
5	Commands	15
5.1	Re-running deletions and anonymisations	15
5.2	Anonymising all personal data	15
6	The admin site	17
6.1	Bulk anonymisation	17
6.2	Personal data tool	17
7	Upgrading	19
7.1	Instructions	19
7.2	Changelog	19
8	Contributing	21
8.1	Installing	21
8.2	Testing	21
8.3	Code overview	22
8.4	Known limitations	22
8.5	Roadmap	22

Tools to help manage user data in the age of GDPR:

- Find, export and anonymise personal data to comply with GDPR requests
- Track anonymisation and deletion of personal data to replay after restoring backups
- Anonymise all models to sanitise working copies of a production database

Install with:

```
pip install django-gdpr-assist
```

Add to your project's `settings.py`:

```
# Add the app
INSTALLED_APPS = (
    ...
    'gdpr_assist',
    ...
)

# Add a new database to log GDPR actions
DATABASES = {
    ...
    'gdpr_log': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': os.path.join(BASE_DIR, 'gdpr-log.sqlite3'),
    },
}
DATABASE_ROUTERS = ['gdpr_assist.routers.EventLogRouter']
```

You'll then need to migrate the new database:

```
./manage.py migrate --database=gdpr_log
```

1.1 Django settings

In addition to the required changes to your settings listed above, there are additional optional settings which you can define to override default behaviour:

1.1.1 `GDPR_PRIVACY_CLASS_NAME = 'PrivacyMeta'`

This allows you to override the default name of the privacy meta class on models.

1.1.2 `GDPR_PRIVACY_INSTANCE_NAME = '_privacy_meta'`

This allows you to override the default name of the instantiated privacy meta class on models.

1.1.3 `GDPR_LOG_DATABASE_NAME = 'gdpr_log'`

The internal name of the log database. You'll need to use this in the `DATABASES` settings, and when migrating.

1.1.4 `GDPR_CAN_ANONYMISE_DATABASE = False`

Set this to `True` to enable the `anonymise_db` management command. You will want this to be `False` on your production deployment.

2.1 Configure your models

Define privacy settings in a `PrivacyMeta` class on your model:

```
class MyModel(models.Model):
    user = models.ForeignKey(
        settings.AUTH_USER_MODEL,
        blank=True,
        null=True,
        on_delete=gdpr_assist.ANONYMISE(models.SET_NULL),
    )
    display_name = models.CharField(max_length=255)
    private_data = models.IntegerField()
    public_data = models.TextField()

    class PrivacyMeta:
        fields = ['display_name', 'private_data']

        def anonymise_private_data(self, instance):
            return 0

        def search(self, value):
            return self.model.objects.filter(display_name__icontains=value)
```

Next:

- See *Privacy Meta* for the full set of `PrivacyMeta` options, and for how to register a third-party model.
- See *Anonymising* for how anonymisation works.
- See *Admin* to register your model in the admin site, and how to use the admin personal data tool to search and export data.

The PrivacyMeta object

A model needs to be registered with `gdpr-assist` in order to use any of the anonymisation or export functionality. The `PrivacyMeta` object tells `gdpr-assist` which fields are private, and what to do with them.

3.1 Registering automatically

If you define a class called `PrivacyMeta` within your model, `gdpr-assist` will automatically detect and register your model.

An instance of your `PrivacyMeta` class will then be available on the attribute `_privacy_meta`, the same way a standard `Meta` class works.

For example:

```
class Comment(models.Model):
    name = models.CharField(max_length=255, blank=True)
    age = models.IntegerField(null=True, blank=True)
    message = models.TextField()

    class PrivacyMeta:
        fields = ['name', 'age']

# The following statements are true:
assert(not hasattr(Comment, 'PrivacyMeta'))
assert(hasattr(Comment, '_privacy_meta'))
assert(Comment._privacy_meta.fields == ['name', 'age'])
```

3.2 Registering manually

Sometimes you will want to define your `PrivacyMeta` class somewhere other than on the model - for example when you want to be able to export or anonymise a third-party object, or if you have a particularly complex privacy meta

definition and want to store it in a separate file for clarity.

The `gdpr_assist.register_model(<ModelClass>, [<PrivacyMetaClass>])` function will let you manually register the model with an optional `PrivacyMeta` class.

For example:

```
from django.contrib.auth.models import User

class UserPrivacyMeta:
    fields = ['first_name', 'last_name', 'email']

gdpr_assist.register_model(User, UserPrivacyMeta)
```

If you omit the privacy meta class, one will be generated for you with the default attributes.

3.3 Attributes

The `PrivacyMeta` object can have the following attributes:

3.3.1 `fields = [...]`

List of the names of fields which contain personal information.

These will be the ones which are anonymised; other fields will be unmodified.

Example:

```
class MyModel(models.Model):
    name = models.CharField(max_length=255)

    class PrivacyMeta:
        fields = ['name']
```

3.3.2 `anonymise_<field_name>(self, instance)`

Custom function to anonymise the named field, for when the standard anonymisers won't produce the desired result. This should also be used for custom field types.

Field name must appear in the `fields` list.

It should not return a value; instead it should operate directly on the instance.

Example:

```
class MyModel(models.Model):
    name = models.CharField(max_length=255)

    class PrivacyMeta:
        def anonymise_name(self, instance):
            instance.name = 'Anon'
```

3.3.3 search_fields = [...]

List of fields to examine when searching for a value in the personal data tool in the admin site.

These field names will be used to build case-insensitive exact matches unless the field name contains a double underscore, __. For example:

- name will create a filter of name__iexact=term
- name__icontains will create a filter of name__icontains=term
- person__name will create a filter of person__name=term

Example:

```
class MyModel(models.Model):
    name = models.CharField(max_length=255)

    class PrivacyMeta:
        search_fields = ['name__icontains']
```

3.3.4 qs = search(self, value)

Function called by the personal data tool in the admin site, to search the model for the value.

The argument self will be a reference to the PrivacyMeta instance.

The default function will use search_fields, but this can be overridden to perform a custom search.

Should return a queryset (or iterable of objects).

Example:

```
class MyModel(models.Model):
    name = models.CharField(max_length=255)

    class PrivacyMeta:
        def search(self, value):
            return self.model.objects.filter(name=value.lower())
```

3.3.5 export_fields = [...]

List of fields to export. By default will export all fields.

Example:

```
class MyModel(models.Model):
    name = models.CharField(max_length=255)

    class PrivacyMeta:
        export_fields = ['name']
```

3.3.6 export_exclude = [...]

List of fields to not export. By default will exclude foreign keys and many to many fields.

If a field is specified in both export_fields and export_exclude, it will be excluded.

Example:

```
class MyModel(models.Model):
    name = models.CharField(max_length=255)
    post_count = models.IntegerField(default=0)

    class PrivacyMeta:
        export_exclude = ['post_count']
```

3.3.7 export_filename = None

The filename to use for this model when exporting records from it. This should include the .csv extension, eg `export_filename = 'user_records.csv'`

If not set, it will default to `<app_name>.<object_name>.csv`, eg `my_app.MyModel.csv`

Example:

```
class MyModel(models.Model):
    name = models.CharField(max_length=255)

    class PrivacyMeta:
        export_filename = 'my_model.csv'
```

3.3.8 data = export(self, instance)

Function called by the personal data tool in the admin site, to export a model instance.

By default will export all fields specified in `export_fields` and not excluded by `export_exclude`. They will all be cast to a string.

The default exporter cannot export foreign keys or many to many fields.

Should return a dict.

Example:

```
class MyModel(models.Model):
    name = models.CharField(max_length=255)

    class PrivacyMeta:
        def export(self, instance):
            return {
                'name': instance.name,
                'lower': instance.lower(),
            }
```

4.1 Model changes

Once a model has been registered with `gdpr-assist`, its instances will have some additional attributes:

4.1.1 `obj.anonymised = BooleanField()`

This is a boolean value stored in the database to register whether the object has been anonymised or not.

4.1.2 `obj.anonymise()`

Call this to anonymise the private fields on the object.

4.2 How anonymisation works

If a field is nullable, the value will be set to `None` (or in the case of blankable strings, `' '`).

If a field is not nullable, the value will be set to a sensible default:

- Numbers will be set to `0`
- Strings will be set to a string representation of the primary key field
- Booleans will be set to `False` (although `NullBooleanField` will always be nullable)
- `DateField` and `DateTimeField` will be set to the current date and time
- `TimeField` will be set to `00:00`
- `DurationField` will be set to `timedelta(0)`
- `EmailField` will be anonymised to `{pk}@anon.example.com`

- URLField will be anonymised to {pk}@anon.example.com
- GenericIPAddressField will be set to 0.0.0.0
- UUIDField will be set to {00000000-0000-0000-0000-000000000000}

These default actions can be overridden by defining a custom anonymiser as `anonymise_<field_name>` method on the `PrivacyMeta` class - see the [PrivacyMeta](#) documentation for more details.

Custom field types will also need a custom anonymiser to be defined.

Some fields cannot be anonymised unless they can be null, and trying to anonymise them without a custom anonymiser will raise a `gdpr_assist.AnonymiseError` exception:

- File fields (`FilePathField`, `FileField`, `ImageField`)
- Relationships (`OneToOneField`, `ForeignKey`)

To ensure data integrity, trying to anonymise a `ManyToManyField` will always raise a `gdpr_assist.AnonymiseError`, unless you are using a custom anonymiser for that field.

The anonymiser cannot anonymise the primary key.

4.3 Anonymising related objects

When anonymising an object, its related objects will not be anonymised automatically. This is to prevent unintentional side-effects.

4.3.1 Using signals

To cascade an anonymisation to another model, use signals:

`gdpr_assist.signals.pre_anonymise`

Called before the object is anonymised. You will be passed the original object.

Example:

```
from gdpr_assist.signals import pre_anonymise

@receiver(pre_anonymise, sender=MyModel)
def anonymise_related(sender, instance):
    instance.my_related_obj.anonymise()
```

`gdpr_assist.signals.post_anonymise`

Called after anonymisation of the object. You will be passed the anonymised object.

Example:

```
from gdpr_assist.signals import pre_anonymise

@receiver(pre_anonymise, sender=MyModel)
def anonymise_related(sender, instance):
    instance.my_related_obj.anonymise()
```


4.3.2 When deleting an object

When an object is deleted, any objects related to the object as `on_delete=gdpr_assist.ANONYMISE(..)` will be anonymised. This takes one argument - the type of `on_delete` to perform on the field itself; for example:

```
class MyModel(models.Model):
    fk = models.ForeignKey(
        TargetModel,
        on_delete=gdpr_assist.ANONYMISE(models.SET_NULL),
    )
```

`ANONYMISE(..)` cannot take `CASCADE` or `PROTECTED` as arguments.

Deleting querysets

gdpr-assist modifies the queryset of registered models so that a bulk deletion will anonymise any related objects which use `ANONYMISE(..)`.

Note that Django does not send delete signals for bulk delete operations in other for situations, so to anonymise related objects when a queryset is deleted, make sure the model being deleted is registered with gdpr-assist.

5.1 Re-running deletions and anonymisations

To re-run a set of deletions and anonymisations, make sure your log database is available, then run:

```
./manage.py gdpr_rerun
```

5.2 Anonymising all personal data

To anonymise all data in all models registered with gdpr-assist:

```
./manage.py anonymise_db
```

This will anonymise all data in the database,

This command can be useful when working on a stage or local copy of the live database. Because it is probably a bad idea to run this on a production database, you will need to enable this command with the setting `GDPR_CAN_ANONYMISE_DATABASE = True`.

6.1 Bulk anonymisation

To add an “Anonymise” option to the actions list for a `ModelAdmin`, subclass `gdpr_assist.admin.ModelAdmin`:

```
import gdpr_assist
class MyAdmin(gdpr_assist.admin.ModelAdmin):
    ...
admin.site.register(MyModel, MyAdmin)
```

6.2 Personal data tool

In the admin site, under `GDPR`, select `Personal data`. This is a tool which lets you find, export, delete and anonymise personal data.

Submitting the search will call the `PrivacyMeta.search()` method on all models registered with `gdpr-assist`.

From there, records can be selected for export, anonymisation or deletion.

This tool is only available to superusers.

For an overview of what has changed between versions, see the *Changelog*.

7.1 Instructions

There are no upgrade instructions yet.

7.2 Changelog

7.2.1 1.0.1, 2018-10-23

Bugfix:

- Managers on registered models which set `use_in_migrations` can now be serialised for migrations.

7.2.2 1.0.0, 2018-09-16

Initial public release

Contributions are welcome by pull request. Check the github issues and project [roadmap](#) to see what needs work.

8.1 Installing

The easiest way to work on GDPR-assist is to fork the project on github, then install it to a virtualenv:

```
virtualenv django-gdpr-assist
cd django-gdpr-assist
source bin/activate
pip install -e git+git@github.com:USERNAME/django-gdpr-assist.git#egg=django-gdpr-
↪assist[dev]
```

(replacing USERNAME with your username).

This will install the development dependencies too, and you'll find the source ready for you to work on in the `src` folder of your virtualenv.

8.2 Testing

Contributions will be merged more quickly if they are provided with unit tests.

Use `setup.py` to run the python tests on your current python environment; you can optionally specify which test to run:

```
python setup.py test [tests[.test_set.TestClass]]
```

Use `tox` to run them on one or more supported versions:

```
tox [-e py36-django1.11] [tests[.test_module.TestClass]]
```

Tox will also generate a coverage HTML report.

You can also use `detox` to run the tests concurrently, although you will need to run `tox -e report` again afterwards to generate the coverage report.

To use a different database (mysql, postgres etc) use the environment variables `DATABASE_ENGINE`, `DATABASE_NAME`, `DATABASE_USER`, `DATABASE_PASSWORD`, `DATABASE_HOST` and `DATABASE_PORT`, eg:

```
DATABASE_ENGINE=pgsql DATABASE_NAME=gdpr_assist_test [...] tox
```

8.3 Code overview

The `handlers.register_model` handler watches for new model definitions which include a `PrivacyMeta` attribute. These models are then registered automatically with the `registry.registry`.

Registration casts and instantiates the `PrivacyMeta` and stores it on the `_privacy_meta` attribute of the model. It also changes the base class of the model to `models.PrivacyModel`, its manager to `models.PrivacyManager` and its queryset to `models.PrivacyQuerySet` to add the necessary anonymisation attributes and methods.

Once all models are registered, `apps.GdprAppConfig.ready` looks at all registered models for a `OneToOneField` or `ForeignKey` which have `on_delete=ANONYMISE(..)`, and then logs the related models with the `registry` so that `handlers.handle_pre_delete` knows to watch them.

When a registered object is deleted, its details are logged to `models.EventLog`, stored in a separate database.

Anonymisation starts with `models.PrivacyModel.anonymise`, which then calls the field-specific anonymise functions in the `PrivacyMeta` instance; fields which do not have one defined use `anonymiser.anonymise_field`.

8.4 Known limitations

- QuerySet bulk deletions on a model will not be detected unless it has a `PrivacyMeta` or is manually registered with `gdpr_assist.register`
- Operations involving gdpr-assist may be slower than normal (ie bulk deletions) due to the additional processing required.

8.5 Roadmap

Features planned for future releases:

- Settings to customise the anonymised field name and `anonymise()` method name on registered models - see *Anonymising objects*
- Subclass the queryset of `on_delete=ANONYMISE(..)` related models which aren't registered, so that bulk deletion always results in anonymisation - see *Anonymising objects*
- Ability to change a relationship field on a registered third-party model to use `on_delete=ANONYMISE(..)`
- A generic view to allow self-service data export, ready to be added to user-facing profile management.
- A generic view to allow self-service data removal, ready to be added to user-facing profile management.

This app does not currently attempt to provide any sort of framework for managing opt-in or consent, because in our experience no two sites are similar enough for a generic solution.