django-rest-metadata-demo

October 15, 2020

1 Generate a RestAPI for Metadata with Django

Philipp S. Sommer

Helmholtz Coastal Data Center (HCDC) Helmholtz-Zentrum Geesthacht, Institute of Coastal Research

1.1 What's the problem with CSW?

- it's XML!
- too complicated to use for scientists or web development

→ Provide metadata through RestAPI

1.1.1 Examples:

- CERA (DKRZ)
- O2A (AWI)

1.2 What is a RestAPI

- official definition at restfulapi.net
- common implementation:
  - provide JSON-formatted data (or other formats) via `GET` request
  - alter data on the server via `PUT` request
  - do something via `POST` request

1.3 Purpose of this talk

Provide a simple and basic example from scratch to show the functionality of serving metadata via RestAPI.

1.3.1 Requirements to run this notebook

- `linux` or `osx`
- `django`
- `djangorestframework`, for the rest api
- `uritemplate`, for generating an openAPI schema

and for generating a graph of the database_ - `graphviz` - `django-extensions`

Install everything via:

```
conda create -n django-metadata -c conda-forge django-extensions graphviz uritemplate djangores
```

[1]:
```
rm -r django-metadata-api
```

1.4 Initialize the django-metadata-api project

[2]:
```
!mkdir django-metadata-api
!django-admin startproject django_metadata_api django-metadata-api
```

[3]:
```
cd django-metadata-api
```

```
/home/psommer/Documents/code/docs/django-rest-metadatenportal/django-metadata-
api
```

1.4.1 Create a django app in this project

We call it `api`. Here we will do all our work.

[4]:
```
!python manage.py startapp api
```

1.5 Make the first migration

This will generate an sqlite3 database (but we could also use something else...)

[5]:
```
!python manage.py migrate
```

```
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, sessions
Running migrations:
  Applying contenttypes.0001_initial… OK
  Applying auth.0001_initial… OK
  Applying admin.0001_initial… OK
  Applying admin.0002_logentry_remove_auto_add… OK
  Applying admin.0003_logentry_add_action_flag_choices… OK
  Applying contenttypes.0002_remove_content_type_name… OK
  Applying auth.0002_alter_permission_name_max_length… OK
  Applying auth.0003_alter_user_email_max_length… OK
  Applying auth.0004_alter_user_username_opts… OK
  Applying auth.0005_alter_user_last_login_null… OK
  Applying auth.0006_require_contenttypes_0002… OK
  Applying auth.0007_alter_validators_add_error_messages… OK
  Applying auth.0008_alter_user_username_max_length… OK
  Applying auth.0009_alter_user_last_name_max_length… OK
  Applying auth.0010_alter_group_name_max_length… OK
  Applying auth.0011_update_proxy_permissions… OK
  Applying auth.0012_alter_user_first_name_max_length… OK
  Applying sessions.0001_initial… OK
```

1.6 The django project structure

[6]:
```
!tree
```

```
.
  api
      admin.py
      apps.py
      __init__.py
      migrations
          __init__.py
      models.py
      tests.py
      views.py
  db.sqlite3
  django_metadata_api
      asgi.py
      __init__.py
      __pycache__
          __init__.cpython-37.pyc
          settings.cpython-37.pyc
          urls.cpython-37.pyc
      settings.py
      urls.py
      wsgi.py
  manage.py

4 directories, 17 files
```

1.7 The project settings

[7]:
```
!cat django_metadata_api/settings.py
```

```
"""
Django settings for django_metadata_api project.

Generated by 'django-admin startproject' using Django 3.1.

For more information on this file, see
https://docs.djangoproject.com/en/3.1/topics/settings/

For the full list of settings and their values, see
https://docs.djangoproject.com/en/3.1/ref/settings/
"""

from pathlib import Path

# Build paths inside the project like this: BASE_DIR / 'subdir'.
```

```
BASE_DIR = Path(__file__).resolve(strict=True).parent.parent


# Quick-start development settings - unsuitable for production
# See https://docs.djangoproject.com/en/3.1/howto/deployment/checklist/

# SECURITY WARNING: keep the secret key used in production secret!
SECRET_KEY = '^)%kc%==1^3l=87*2!+!nmch#jgm$#y#ad_-i=zg__w1*^8wf*'

# SECURITY WARNING: don't run with debug turned on in production!
DEBUG = True

ALLOWED_HOSTS = []


# Application definition

INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
]

MIDDLEWARE = [
    'django.middleware.security.SecurityMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.common.CommonMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
    'django.middleware.clickjacking.XFrameOptionsMiddleware',
]

ROOT_URLCONF = 'django_metadata_api.urls'

TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [],
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
                'django.contrib.messages.context_processors.messages',
            ],
        },
    },
]

WSGI_APPLICATION = 'django_metadata_api.wsgi.application'


# Database
# https://docs.djangoproject.com/en/3.1/ref/settings/#databases

DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': BASE_DIR / 'db.sqlite3',
    }
}


# Password validation
# https://docs.djangoproject.com/en/3.1/ref/settings/#auth-password-validators

AUTH_PASSWORD_VALIDATORS = [
    {
        'NAME':
'django.contrib.auth.password_validation.UserAttributeSimilarityValidator',
    },
    {
        'NAME':
'django.contrib.auth.password_validation.MinimumLengthValidator',
    },
    {
        'NAME':
'django.contrib.auth.password_validation.CommonPasswordValidator',
    },
    {
        'NAME':
'django.contrib.auth.password_validation.NumericPasswordValidator',
    },
]


# Internationalization
# https://docs.djangoproject.com/en/3.1/topics/i18n/

LANGUAGE_CODE = 'en-us'
```

```
TIME_ZONE = 'UTC'

USE_I18N = True

USE_L10N = True

USE_TZ = True


# Static files (CSS, JavaScript, Images)
# https://docs.djangoproject.com/en/3.1/howto/static-files/

STATIC_URL = '/static/'
```

1.8 Add the necessary apps to the settings

[8]:
```
%%writefile -a django_metadata_api/settings.py

INSTALLED_APPS += [
    "api",
    "rest_framework",
    "django_extensions",
]
```

```
Appending to django_metadata_api/settings.py
```

1.9 Django models

Each `Model` (inherits the `django.db.models.Model` class) defines a table in our (sqlite3) database. The `fields` of each model correspond to the columns of the database table.

Initially, there are no models defined.

[9]:
```
!cat api/models.py
```

```
from django.db import models

# Create your models here.
```

1.10 Creating models

So let's define some.

[10]:
```
%%writefile api/models.py

from django.db import models


class Institution(models.Model):
    """A research institution."""

    name = models.CharField(
        max_length=250,
        help_text="Name of the institution",
    )

    abbreviation = models.CharField(
        max_length=10,
        help_text="Abbreviation of the institution"
    )

    def __str__(self):
        return f"{self.name} ({self.abbreviation})"
```

```
Overwriting api/models.py
```

1.11 Django models

and some more

[11]:
```
%%writefile -a api/models.py

class Person(models.Model):
    """A person."""

    first_name = models.CharField(
        max_length=50,
        help_text="First name of the person"
    )

    last_name = models.CharField(
        max_length=50,
        help_text="Last name of the person"
    )

    email = models.EmailField(
        max_length=255,
        help_text="Email address of the person.",
    )

    institution = models.ForeignKey(
        Institution,
        on_delete=models.PROTECT,
        help_text="Research institution of the person."
    )

    def __str__(self):
        return f"{self.first_name} {self.last_name} ({self.institution.
↪abbreviation})"
```

```
Appending to api/models.py
```

1.12 Django models

and some more

[12]:
```
%%writefile -a api/models.py

class Project(models.Model):
    """A research project."""

    name = models.CharField(
        max_length=250,
        help_text="Full name of the project",
    )

    abbreviation = models.CharField(
        max_length=50,
        help_text="Abbreviation of the project."
    )

    pi = models.ForeignKey(
        Person,
        on_delete=models.PROTECT,
        help_text="Principal investigator of the model."
    )

    def __str__(self):
        return f"{self.name} ({self.abbreviation})"
```

```
Appending to api/models.py
```

1.13 Django models

and some more

[13]:
```
%%writefile -a api/models.py

class Dataset(models.Model):
    """A dataset output of a model."""

    class DataSource(models.TextChoices):
        """Available data sources."""

        model = "MODEL", "derived from a climate model"
        satellite = "SATELLITE", "derived from satellite observation"
    name = models.CharField(
        max_length=200,
        help_text="Name of the dataset."
    )

    source_type = models.CharField(
        max_length=20,
        choices=DataSource.choices,
        help_text="How the data has been derived."
    )

    project = models.ForeignKey(
        Project,
        on_delete=models.CASCADE,
        help_text="The project this dataset belongs to."
    )

    contact = models.ForeignKey(
        Person,
        on_delete=models.PROTECT,
        help_text="The contact person for this dataset",
    )

    def __str__(self):
        return f"{self.name} ({self.project.abbreviation})"
```

```
Appending to api/models.py
```

1.14 Django models

and some more

[14]:
```
%%writefile -a api/models.py

class Parameter(models.Model):
    """A standardized parameter in our database."""

    name = models.CharField(
        max_length=200,
        help_text="Name of the dataset."
    )

    unit = models.CharField(
        max_length=20,
        help_text="Units of the parameter."
    )

    long_name = models.CharField(
        max_length=250,
        help_text="Description of the parameter"
    )

    dataset = models.ForeignKey(
        Dataset,
        help_text="The dataset that contains this parameter",
        related_name="parameters",
        on_delete=models.CASCADE,
    )

    def __str__(self):
        return f"{self.name} ({self.unit})"
```

```
Appending to api/models.py
```

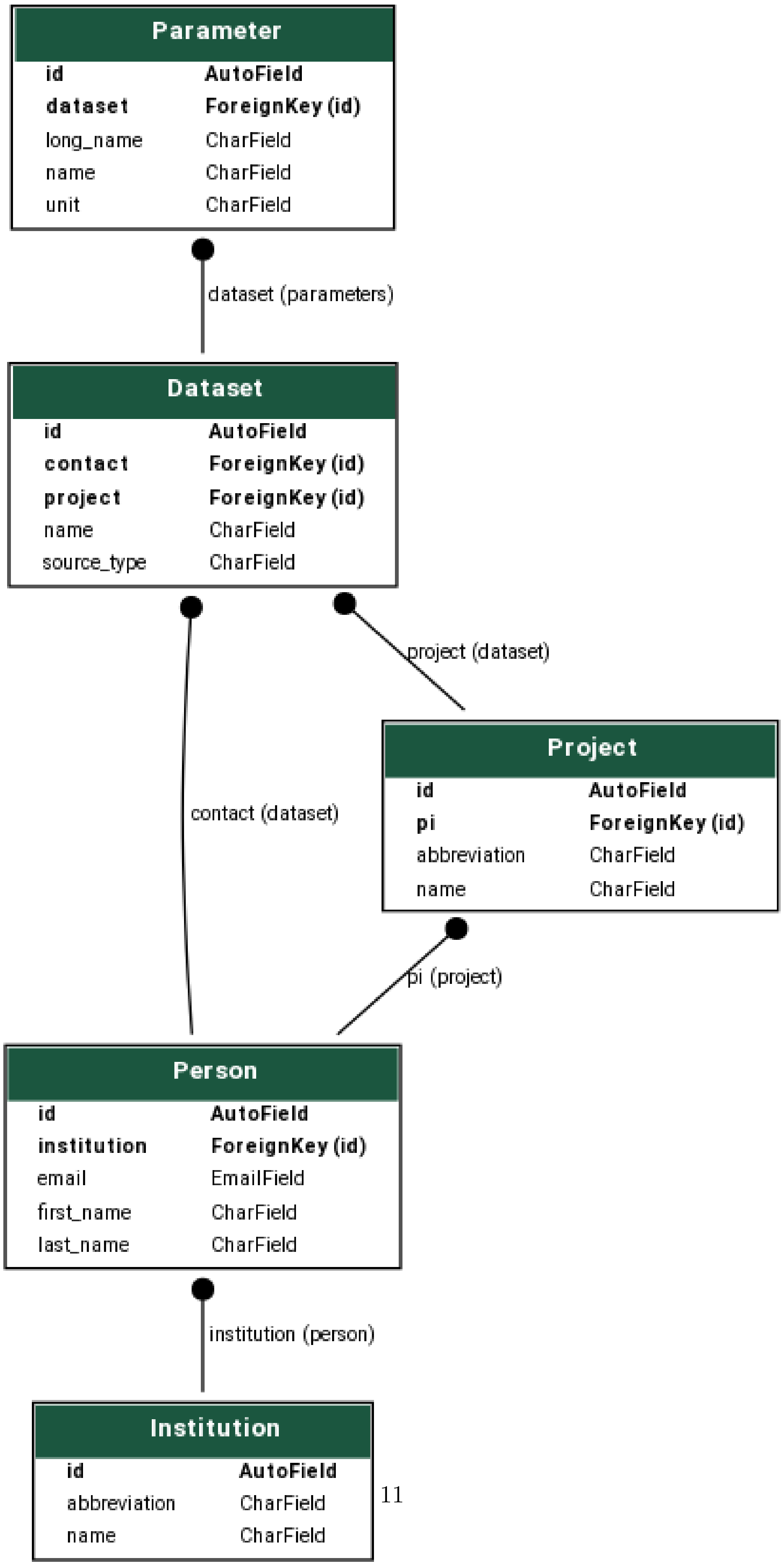
1.15 Getting an overview

django-extensions provide the functionality to show a graph of our models. So let's do this

[15]:
```
!python manage.py graph_models api > apigraph.dot
!dot apigraph.dot -Tpng -o apigraph.png
```

[16]:
```
from IPython.display import Image
Image(filename="apigraph.png")
```

[16]:


Parameter
id AutoField
dataset ForeignKey (id)
long_name CharField
name CharField
unit CharField
dataset (parameters)
Dataset
id AutoField
contact ForeignKey (id)
project ForeignKey (id)
name CharField
source_type CharField
project (dataset)
contact (dataset)
Project
id AutoField
pi ForeignKey (id)
abbreviation CharField
name CharField
pi (project)
Person
id AutoField
institution ForeignKey (id)
email EmailField
first_name CharField
last_name CharField
institution (person)
Institution
id AutoField
abbreviation CharField
name CharField

1.16 Update the database

So far, we just wrote some python. Now tell Django to register our models in the (sqlite3) database:

[17]:
```
!python manage.py makemigrations  # creates the migration scripts
```

```
Migrations for 'api':
  api/migrations/0001_initial.py
    - Create model Dataset
    - Create model Institution
    - Create model Person
    - Create model Project
    - Create model Parameter
    - Add field contact to dataset
    - Add field project to dataset
```

[18]:
```
!python manage.py migrate  # creates the tables in the database
```

```
Operations to perform:
  Apply all migrations: admin, api, auth, contenttypes, sessions
Running migrations:
  Applying api.0001_initial… OK
```

1.17 Add serializers to our models

A serializer transforms your model into JSON (and more).

[19]:
```
%%writefile api/serializers.py

from rest_framework import serializers
from api import models


class InstitutionSerializer(serializers.HyperlinkedModelSerializer):

    class Meta:
        model = models.Institution
        fields = '__all__'
```

```
Writing api/serializers.py
```

1.18 And serializers for the other models

[20]:
```
%%writefile -a api/serializers.py

class PersonSerializer(serializers.HyperlinkedModelSerializer):

    class Meta:
        model = models.Person
        fields = '__all__'


class ProjectSerializer(serializers.HyperlinkedModelSerializer):

    class Meta:
        model = models.Project
        fields = '__all__'


class DatasetSerializer(serializers.HyperlinkedModelSerializer):

    class Meta:
        model = models.Dataset
        fields = '__all__'


class ParameterSerializer(serializers.HyperlinkedModelSerializer):

    class Meta:
        model = models.Parameter
        fields = '__all__'
```

```
Appending to api/serializers.py
```

1.19 Generate the viewset for the models

A viewset (comparable to an HTML webpage) tells django, you to display and update the serialized models.

[21]:
```
%%writefile api/views.py

from rest_framework import viewsets
from rest_framework import permissions

from api import models, serializers


class InstitutionViewSet(viewsets.ModelViewSet):
    """View the institutions"""
    queryset = models.Institution.objects.all()
    serializer_class = serializers.InstitutionSerializer
```

```
Overwriting api/views.py
```

1.20 And viewsets for the other models

[22]:
```
%%writefile -a api/views.py

class PersonViewSet(viewsets.ModelViewSet):
    """View the institutions"""

    queryset = models.Person.objects.all()
    serializer_class = serializers.PersonSerializer


class ProjectViewSet(viewsets.ModelViewSet):
    """View the institutions"""

    queryset = models.Project.objects.all()
    serializer_class = serializers.ProjectSerializer


class DatasetViewSet(viewsets.ModelViewSet):
    """View the institutions"""

    queryset = models.Dataset.objects.all()
    serializer_class = serializers.DatasetSerializer


class ParameterViewSet(viewsets.ModelViewSet):
    """View the institutions"""

    queryset = models.Parameter.objects.all()
    serializer_class = serializers.ParameterSerializer
```

```
Appending to api/views.py
```

1.21 Define the router

We generated the webpages, but did not tell anything about where to find them.

This is the job of the router.

[23]:
```
%%writefile api/urls.py

from django.urls import include, path
```

```python
from rest_framework import routers
from api import views

router = routers.DefaultRouter()
router.register(r'institutions', views.InstitutionViewSet)
router.register(r'persons', views.PersonViewSet)
router.register(r'projects', views.ProjectViewSet)
router.register(r'datasets', views.DatasetViewSet)
router.register(r'parameters', views.ParameterViewSet)

# Wire up our API using automatic URL routing.
# Additionally, we include login URLs for the browsable API.
urlpatterns = [
    path('', include(router.urls)),
]
```

```
Writing api/urls.py
```

1.22 Add our api app to the main router file

We now need to add the urls of our API to the main project.

[24]:
```
cat django_metadata_api/urls.py
```

```
"""django_metadata_api URL Configuration

The `urlpatterns` list routes URLs to views. For more information please see:
    https://docs.djangoproject.com/en/3.1/topics/http/urls/
Examples:
Function views
    1. Add an import:  from my_app import views
    2. Add a URL to urlpatterns:  path('', views.home, name='home')
Class-based views
    1. Add an import:  from other_app.views import Home
    2. Add a URL to urlpatterns:  path('', Home.as_view(), name='home')
Including another URLconf
    1. Import the include() function: from django.urls import include, path
    2. Add a URL to urlpatterns:  path('blog/', include('blog.urls'))
"""
from django.contrib import admin
from django.urls import path

urlpatterns = [
    path('admin/', admin.site.urls),
]
```

1.23 Add our api urls

[25]:
```
%%writefile -a django_metadata_api/urls.py

from django.urls import include

urlpatterns.append(path('', include("api.urls")))
```

```
Appending to django_metadata_api/urls.py
```

1.24 Starting django

Now run

python manage.py runserver

in an external terminal to start the development server and head over to http://127.0.0.1:8000

1.25 Add the parameters to the dataset

[26]:
```
%%writefile -a api/serializers.py


class DatasetSerializer(serializers.HyperlinkedModelSerializer):

    parameters = ParameterSerializer(many=True)

    class Meta:
        model = models.Dataset
        fields = '__all__'
```

```
Appending to api/serializers.py
```

Checkout the changes at http://127.0.0.1:8000/datasets

1.26 Enable the admin interface

[27]:
```
!cat api/admin.py
```

```
from django.contrib import admin

# Register your models here.
```

[28]:
```
%%writefile api/admin.py

from django.contrib import admin
from api import models
```

```
class ParameterInline(admin.TabularInline):
    model = models.Parameter


@admin.register(models.Dataset)
class DatasetAdmin(admin.ModelAdmin):
    """Administration class for the :model:`api.Dataset` model."""

    inlines = [ParameterInline]

    search_fields = ["name", "project"]
```

```
Overwriting api/admin.py
```

1.27 Create a user to access the admin interface

Open a terminal and run

`python manage.py createsuperuser --email admin@example.com --username admin`

And checkout `http://127.0.0.1:8000/admin`

1.28 Restrict PUT and POST to authenticated users

Djangos Rest framework comes with a login and logout functionality that we need to insert into our projects urls.py router file.

[29]:
```
%%writefile -a django_metadata_api/urls.py

urlpatterns.insert(
    -2, path('api-auth/', include('rest_framework.urls',␣
 ↪namespace='rest_framework'))
)
```

```
Appending to django_metadata_api/urls.py
```

1.29 Add the permission to our viewsets

[30]:
```
%%writefile -a api/views.py

for view in [PersonViewSet, DatasetViewSet, InstitutionViewSet, ProjectViewSet,␣
 ↪ParameterViewSet]:
    view.permission_classes = [permissions.IsAuthenticatedOrReadOnly]
```

```
Appending to api/views.py
```

Now you'll see that you cannot make POST requests anymore to http://127.0.0.1:8000/datasets (for instance).

Login at http://127.0.0.1:8000/api-auth/login and it will be possible again.

1.30 Export the schema

Now we can export our database schema to show others, how our RestAPI is structured. For this purpose, we add a new view to our api.

[31]:
```
%%writefile -a api/urls.py

from rest_framework.schemas import get_schema_view

urlpatterns.append(
    path('schema', get_schema_view(
        title="Metadata Portal",
        description="API for retrieving metadata",
        version="1.0.0",
        urlconf='api.urls',
    ), name='openapi-schema'),
)
```

```
Appending to api/urls.py
```

Head over to http://127.0.0.1:8000/schema to see the results

1.31 The END

That's it. Now you have a well-defined and functional RestAPI with just a few lines of code!