MyWorkWiki

Apertif Task Database - Project Setup

- https://docs.djangoproject.com/en/2.1/ [https://docs.djangoproject.com/en/2.1/]

Initial Django Setup

setup fresh environment

```
> virtualenv r:\env_atdb
> r:\env_atdb\Scripts\activate (env_atdb.bat)
> python -m pip install --upgrade pip (upgrade to pip-18.0)
```

Install Django

- https://docs.djangoproject.com/en/2.1/intro/tutorial01/ [https://docs.djangoproject.com/en/2.1/intro/tutorial01/]

```
> pip install Django
> pip install -U Django (upgrade to Django 2)
```

Create "atdb" project and "taskdatabase" app

```
> mdir r:\atdb-trunk
> cd r:\atdb-trunk
> django-admin startproject atdb
> cd atdb
> python manage.py startapp taskdatabase
```

Test

- `python manage.py runserver`
- http://localhost:8000/atdb [http://localhost:8000/atdb]

Database

- https://docs.djangoproject.com/en/2.1/intro/tutorial02/ [https://docs.djangoproject.com/en/2.1/intro/tutorial02/]

- get the postgres driver: `pip install psycopg2`
- point the `settings.py` to the following (development) Postgres database:

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql_psycopg2',
```

```
        'USER': 'atdb_admin',
        'PASSWORD': 'atdb123',

        # database runs locally in postgres
        'NAME': 'atdb_trunk',
        'HOST': 'localhost',
        'PORT': '',
    },
}
```

- manually create the above `atdb_trunk` database and `atdb_admin` user in Postgres
- python manage.py migrate

Initial models.py

- ATDB Initial models.py
- register app in settings :

```
INSTALLED_APPS = [
  'taskdatabase.apps.TaskdatabaseConfig',
```

- `python manage.py makemigrations taskdatabase`
- `python manage.py migrate`

Initial views.py

- https://docs.djangoproject.com/en/2.1/intro/tutorial03/ [https://docs.djangoproject.com/en/2.1/intro/tutorial03/]

A basic view:

urls.py

```
urlpatterns = [
    # ex: /atdb/
    path('', views.index, name='index'),

    # ex: /atdb/5/
    path('<int:dataproduct_id>/', views.detail, name='detail'),

]
```

views.py

```
from django.http import HttpResponse

def index(request):
    return HttpResponse("Welcome to ATDB.")

def detail(request, dataproduct_id):
    return HttpResponse("You're looking at dataproduct %s." % dataproduct_id)
```

Admin Site

Create Superuser

- `python manage.py createsuperuser`
    - admin (admin).

- Now the admin screen should be accessible: http://localhost:8000/admin/ [http://localhost:8000/admin/]

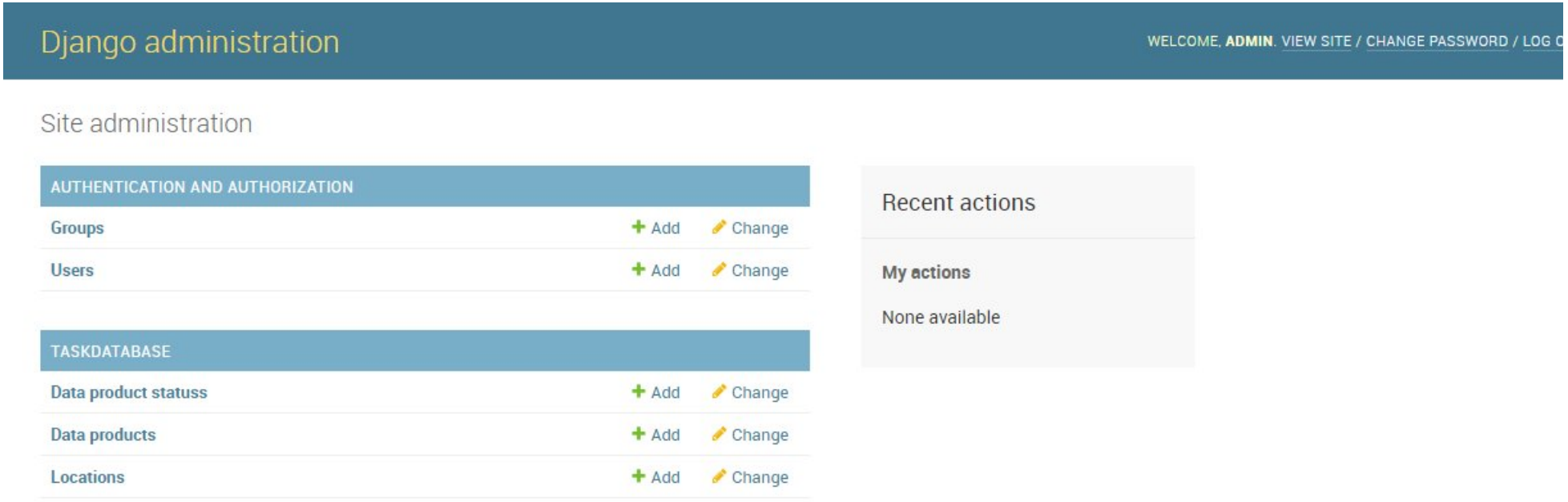
Register models in "admin.py"

Now for the initial model, but this has to be done for every class that is later added to `models.py`

admin.py

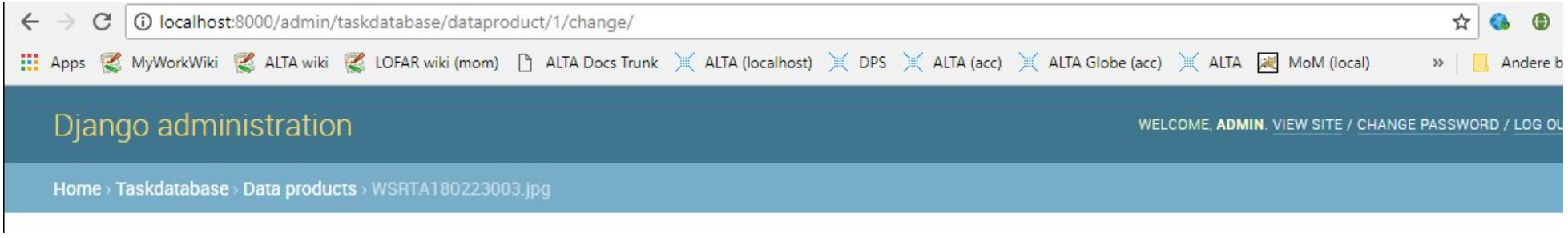
```
from django.contrib import admin
from .models import Location, DataProductStatus, DataProduct

admin.site.register(Location)
admin.site.register(DataProductStatus)
admin.site.register(DataProduct)
```



(initial admin screen)

Now you can add some test data through the admin screen. (tedius, but okay for initial testing).



Change data product

HISTORY

Filename:	WSRTA180223003.jpg
Description:	M51 color image
Type:	image
RunId:	180810001
Timestamp of creation in the database.	Date: 2018-08-10 Today | Time: 09:20:07 Now | Note: You are 2 hours ahead of server time.
Size:	1000
Quality:	good
Location:	localhost/alta-static/180223003
Actual state:	created
State history:	defined

Delete | Save and add another | Save and continue editing | SAVE

Advanced Setup

Django Rest Framework

Follow the steps at http://www.django-rest-framework.org/ [http://www.django-rest-framework.org/]

Note that we are now using Django 2.0, which has a slightly different url syntax as described in the DRF documentation. The root urls.py should now look like this:

atdb\urls.py

```
urlpatterns = [
    path('atdb/', include('taskdatabase.urls')),
    path('admin/', admin.site.urls),
    # url(r'^api-auth/', include('rest_framework.urls')), # old DRF syntax
    path('api-auth/', include('rest_framework.urls')),
]
```

Add a `serializer.py` containing the first basic serializer:

serializers.py

```
from rest_framework import serializers
from .models import DataProduct

class DataProductSerializer(serializers.ModelSerializer):

    class Meta:
        model = DataProduct
        fields = '__all__'
```

Switch to Class Based Generic Views

- http://www.django-rest-framework.org/api-guide/generic-views/#generic-views [http://www.django-rest-framework.org/api-guide/generic-views/#generic-views]

urls.py

```
from django.urls import path

from .views import index, DataProductListView, DataProductDetailsView

urlpatterns = [
    # ex: /atdb/
    path('', index, name='index'),

    # ex: /atdb/dataproducts/
    path('dataproducts/', DataProductListView.as_view()),

    # ex: /atdb/dataproducts/5/
    path('dataproducts/<int:dataproduct_id>/', DataProductDetailsView.as_view()),
]
```

views.py

```
from django.http import HttpResponse
from rest_framework import generics

from .models import DataProduct
from .serializers import DataProductSerializer

# --- class based views ---
class DataProductListView(generics.ListCreateAPIView):
    model = DataProduct
    queryset = DataProduct.objects.all()
    serializer_class = DataProductSerializer

class DataProductDetailsView(generics.RetrieveUpdateDestroyAPIView):
    model = DataProduct
    queryset = DataProduct.objects.all()
    serializer_class = DataProductSerializer
```

Now the REST API works (when you have the app running):

- http://localhost:8000/atdb/dataproducts [http://localhost:8000/atdb/dataproducts]
- http://localhost:8000/atdb/dataproducts/1/ [http://localhost:8000/atdb/dataproducts/1/]

localhost:8000/atdb/dataproducts/

Apps MyWorkWiki ALTA wiki LOFAR wiki (mom) ALTA Docs Trunk ALTA (localhost) DPS ALTA (acc) ALTA Globe (acc) ALTA MoM (local) » Andere bla

Django REST framework admin

Data Product List

Data Product List

OPTIONS GET

```
GET /atdb/dataproducts/
```

```
HTTP 200 OK
Allow: GET, POST, HEAD, OPTIONS
Content-Type: application/json
Vary: Accept

[
    {
        "id": 1,
        "filename": "WSRTA180223003.jpg",
        "description": "M51 color image",
        "type": "image",
        "taskId": "180810001",
        "creationTime": "2018-08-10T09:20:07Z",
        "size": 1000,
        "quality": "good",
        "location": 1,
        "actual_state": 2,
        "state_history": 1
    },
    {
        "id": 2,
        "filename": "WSRTA180223003_ALL_IMAGE.jpg",
        "description": "unknown",
        "type": "image",
        "taskId": "180223003_IMG",
        "creationTime": "2018-08-10T10:23:42Z",
        "size": 123,
        "quality": "okay",
        "location": 2,
        "actual_state": 2,
        "state_history": 1
    }
]
```

Raw data HTML form

Add filters

Django filters come with DRF and makes it easy to do queries on the REST API using url variables.

- https://django-filter.readthedocs.io/en/latest/index.html [https://django-filter.readthedocs.io/en/latest/index.html]

settings.py

```
# settings.py
INSTALLED_APPS = [
    ...
    'rest_framework',
    'django_filters',
]

REST_FRAMEWORK = {
    'DEFAULT_FILTER_BACKENDS': (
        'django_filters.rest_framework.DjangoFilterBackend',
        ...
    ),
}
```

views.py

```
class DataProductFilter(filters.FilterSet):

    class Meta:
        model = DataProduct

        fields = {
            'type': ['exact', 'in'],  # ../dataproducts?dataProductType=IMAGE,VISIBILITY
            'description': ['exact', 'icontains'],
            'taskId': ['exact', 'icontains'],
            'creationTime': ['gt', 'lt', 'gte', 'lte', 'contains', 'exact']
        }

        class DataProductListView(generics.ListCreateAPIView):
    model = DataProduct
    queryset = DataProduct.objects.all()
    serializer_class = DataProductSerializer

    # using the Django Filter Backend - https://django-filter.readthedocs.io/en/latest/index.html
    filter_backends = (filters.DjangoFilterBackend,)
    filter_class = DataProductFilter
```

This an example of how a filter now returns a result in the REST API:

Django REST framework admin

Data Product List

Data Product List

Filters OPTIONS GET

```
GET /atdb/dataproducts/?description__icontains=M51
```

```
HTTP 200 OK
Allow: GET, POST, HEAD, OPTIONS
Content-Type: application/json
Vary: Accept

[
    {
        "id": 1,
        "filename": "WSRTA180223003.jpg",
        "description": "M51 color image",
        "type": "image",
        "taskId": "180810001",
        "creationTime": "2018-08-10T09:20:07Z",
        "size": 1000,
        "quality": "good",
        "location": 1,
        "actual_state": 2,
        "state_history": 1
    }
]
```

VCS (git & svn)

Later the source will (probably) have to be integrated with the Apertif subversion repository, but for the initial 'prototype phase' I will use git/github for a quicker and less restricted way of working.

- https://github.com/vermaas/atdb [https://github.com/vermaas/atdb]
