記事
Toshihiko Minamoto · 2022年12月21日 7m read

Open Exchange

Django 入門 パート 3

Django の可能性と IRIS の使用方法を引き続き観察しています。 初めにモデルの定義方法と、IRIS に存在しているテーブルへの接続方法を確認し、次に組み込みの Django 管理ポータルを拡張して、モデルに含まれるデータの表示、フィルタ、編集、そしてページネーションの機能を追加しました。

では、実際の動作を確認しましょう。posts-and-tags パッケージで使用したデータで Django に REST API を作成します。

それには、Django REST Framework を使用します。

Django REST Framework は、Web API を構築するための強力で柔軟性を備えたツールキットです。

REST Framework の使用を推奨するのには、以下のような理由があります。

- Web で閲覧可能な API には、開発者のユーザビリティにおいて大きなメリットがあります。
- OAuth1a と OAuth2 の認証ポリシーを含むパッケージ
- ORM と非 ORM データソースの両方をサポートするシリアル化
- すべてをカスタマイズ可能。強力な機能が必要なければ、通常の関数ベースのビューを使用できます。
- 詳細なドキュメントと優れたコミュニティサポート
- Mozilla、Red Hat、Heroku、Eventbrite など、世界的に有名な企業が使用・信頼

まず、依存関係で requirements.txt を更新する必要があります。

```
# Django ????
django>=4.0.0

# Django ? InterSystems IRIS ??????InterSystems ? DB-API ?????
django-iris=>0.1.13
```

```
https://raw.githubusercontent.com/intersystems-community/iris-driver-
distribution/main/DB-API/intersystems_irispython-3.2.0-py3-none-any.whl

# Django REST Framework ?????????????
djangorestframework>=3.4.4
# ????? API ? Markdown ????
markdown>=3.0.0
# ????????
django-filter>=1.0.1
```

そして、これらをインストールします。

```
python install -r requirements.txt
```

API の初稿

urls.py ファイルを以下に更新します。 ここでは、API のルートを ***api/*** に更新し、API リクエストに対し、http://localhost:8000/api/ がルートとして使用されるようにします。

```
from django.contrib import admin
from django.urls import path, include
from rest_framework import routers

router = routers.DefaultRouter()

urlpatterns = [
    path('api/', include(router.urls)),
    path('admin/', admin.site.urls),
    path('api-auth/', include('rest_framework.urls')),
]
```

Django REST Framework には、settings.py の DEBUG=True によりサーバーが開発モードで実行している場合、API の UI が組み込まれています。 この URL を開きましょう。

何も定義されておらず、フレームワークが URL に接続されているだけでも、すべてが機能しています。認証を必要とするリクエストに対しては、認証がサポートされています。

```
$ curl http://127.0.0.1:8000/api/
{}
```

プロジェクトの API を定義しましょう。最低限、REST Framework のいくつかの機能を使用します。

- シリアライザー - クエリセットやモデルインスタンスなどの複雑なデータをネイティブ Python データ型に変換し、JSON や XML などのコンテンツタイプに簡単にレンダリングできるようにします。シリアライザーは逆シリアル化も提供しているため、着信データを検証してから、解析したデータを複雑な型に変換し直すことも可能です。
- ビューセット - 関連する一連のビューのロジックを 1 つのクラスにまとめることができます。

Post 用のエンドポイントを追加しましょう。 とてもシンプルではありますが、 更新された **urls.py _** のコンテンツを見てみましょう。

```
from django.contrib import admin
```

```
from django.urls import path, include
from rest_framework import routers, serializers, viewsets
from .models import CommunityPost

router = routers.DefaultRouter()

class CommunityPostSerializer(serializers.HyperlinkedModelSerializer):
    class Meta:
        # class with model
        model = CommunityPost
        # list of fields to show, or just '__all__'
        fields = '__all__'
# ViewSets define the view behavior.
class CommunityPostViewSet(viewsets.ModelViewSet):
    queryset = CommunityPost.objects.all()
    serializer_class = CommunityPostSerializer

# connect it with API
router.register(r'posts', CommunityPostViewSet)

urlpatterns = [
    path('api/', include(router.urls)),
    path('admin/', admin.site.urls),
    path('api-auth/', include('rest_framework.urls')),
]
```

これで Web UI に表示されるようになりました。

ここのリンクをクリックすると、そのレスポンスを確認できます。

最後までスクロールすると、新しい項目用に生成されたフォームがあります。これは POST リクエストで追加可能です。 すべてのフィールドがプロパティの型に適しています。

項目リストで任意の項目の URL をクリックして、これを確認します。 レスポンスのこの項目と、PUT リクエストを使った編集フォームのみです。

認証

PUT か POST でデータを変更できるようになりました。 認証の要件はまだ有効化されていません。 REST Framework には、使用できる認証の組み合わせが様々用意されているため、匿名アクセスの読み取り専用リソースを一部開放することができます。 そして、変更を行うための認証を行います。
または、アクセスを完全に閉鎖することも可能です。
ここでは、匿名の読み取り専用に構成し、変更には認証を必要とするようにしましょう。
それには、次のコードを ***settings.py*** に追加すれば完了です。

```
REST_FRAMEWORK = {
    # Use Django's standard `django.contrib.auth` permissions,
    # or allow read-only access for unauthenticated users.
    'DEFAULT_PERMISSION_CLASSES': [
        'rest_framework.permissions.DjangoModelPermissionsOrAnonReadOnly',
    ],
}
```

これを使用すれば、Django 管理用に前に作成したユーザー名とパスワードなどでログインするまで、フォームが表示されなくなります。

ページネーション

デフォルトではページネーションはありませんが、リストクエリに簡単に追加できます。 ***settings.py*** の **REST_FRAMEWORK** 変数を更新しましょう。
ページネーションクラスとデフォルトのページサイズをセットアップします。

```
REST_FRAMEWORK = {
...
    'DEFAULT_PAGINATION_CLASS': 'rest_framework.pagination.LimitOffsetPagination',
    'PAGE_SIZE': 10,
...
}
```

これにより、生成される JSON がわずかに変わりました。「次へ」や「前へ」といったページリンクや全項目数などの関連する項目が追加されています。 これで、Web UI でページを移動できるようになりました。

フィルタと検索

フィルタ機能と検索機能の追加も非常に単純です。 ***settings.py*** の **REST_FRAMEWORK** 変数を更新しましょう。

```
REST_FRAMEWORK = {
...
    'DEFAULT_FILTER_BACKENDS': [
        'django_filters.rest_framework.DjangoFilterBackend',
        'rest_framework.filters.SearchFilter',
    ],
...
]
```

そして、CommunityPostViewSet をフィルタと検索用のフィールドのリストで更新します。

```
class CommunityPostViewSet(viewsets.ModelViewSet):
    queryset = CommunityPost.objects.all()
    serializer_class = CommunityPostSerializer
    filterset_fields = ['posttype', 'lang', 'published']
    search_fields = ['name',]
```

これで、Web UI で動作するようになりました。

最後に、完全に機能する REST API が完成しましたが、今のところ、このリソース専用の API です。
非常に単純ですが、十分にカスタマイズ可能で、他のリソースに接続したり、リンクしたリすることが可能です。

#Embedded Python #Python #InterSystems IRIS
InterSystems Open Exchangeで関連アプリケーションを確認してください

ソースURL:
https://jp.community.intersystems.com/post/django-%E5%85%A5%E9%96%80-%E3%83%91%E3%83%BC%E3%83%88-3