The simple
but powerful
elegance of

Django REST Framework

start

Morten Barklund

Full-stack
Freelancer
Rookie Pythonista
Decent Djangoneer

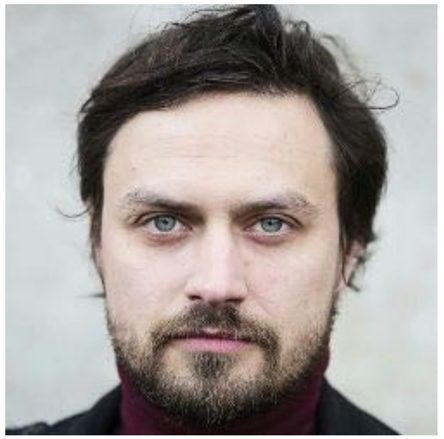


Table of Contents

next

01

Django REST Framework

- JSON-based REST API
- Heavily ORM-backed serialization
- Authentication
- Validation

next

```python
from django.contrib.auth.models import User
from rest_framework import serializers

class UserSerializer(serializers.ModelSerializer):
  class Meta:
    model = User
    fields = ('username', 'email', )
```


next

```
from django.contrib.auth.models import User
from rest_framework import viewsets
from .serializers import UserSerializer

class UserViewSet(viewsets.ModelViewSet):
  queryset = User.objects.all()
  serializer_class = UserSerializer
```


next

```
from django.conf.urls import url, include
from rest_framework import routers
from . import views

router = routers.DefaultRouter()
router.register(r'users', views.UserViewSet)

urlpatterns = [
  url(r'^', include(router.urls)),
  url(r'^api-auth/', include(‘rest_framework.urls',
    namespace='rest_framework'))
]
```


next

```
INSTALLED_APPS = (
  ...
  'rest_framework',
)

REST_FRAMEWORK = {
  'DEFAULT_PERMISSION_CLASSES': (
    'rest_framework.permissions.AllowAny',
  )
}
```

```
GET    /users/
POST   /users/
GET    /users/<pk>/
PUT    /users/<pk>/
PATCH  /users/<pk>/
DELETE /users/<pk>/
```


next

02

Permissions

- Who can do what
- Object collections
- Individual objects

```
class Book(models.Model):
  author = models.ForeignKey('User')
  title = models.TextField()

class BookSerializer(serializers.ModelSerializer):
  class Meta:
    model = Book
    fields = ('author', 'title', )

class BookViewSet(viewsets.ModelViewSet):
  queryset = Book.objects.all()
  serializer_class = BookSerializer

router = routers.DefaultRouter()
router.register(r'books', BookViewSet)
```

```
INSTALLED_APPS = (
  ...
  'dry_rest_permissions',
)
```

```python
from dry_rest_permissions.generics import DRYPermissions

class BookViewSet(viewsets.ModelViewSet):
  permission_classes = (DRYPermissions,)
  queryset = Book.objects.all()
  serializer_class = BookSerializer
```

```
class Book(models.Model):
  author = models.ForeignKey('User')
  ...

  @staticmethod
  def has_read_permission(request):
    return True

  @staticmethod
  def has_write_permission(request):
    return False

  @staticmethod
  @authenticated_users
  def has_create_permission(request):
    return True
```

```
class Book(models.Model):
  ...

  @staticmethod
  @authenticated_users
  def has_write_permission(request):
    return True

  def has_object_write_permission(self, request):
    return request.user == self.author
```

```python
class Book(models.Model):
  ...

  @allow_staff_or_superuser
  def has_object_delete_permission(self, request):
    return request.user == self.author
```

03 Examples

```
class Group(models.Model):
  name = models.CharField(max_length=128)
  is_public = models.BooleanField(default=True)
  members = models.ManyToManyField(“User",
    through='Membership')

class Membership(models.Model):
  user = models.ForeignKey("User")
  group = models.ForeignKey(Group)
  is_admin = models.BooleanField(default=False)
```

```
class GroupSerializer(serializers.ModelSerializer):
  class Meta:
    model = 'group'
    fields = ('name', 'members', )

class GroupViewSet(viewsets.ModelViewSet):
  permission_classes = (DRYPermissions,)
  queryset = Group.objects.all()
  serializer_class = GroupSerializer

router = routers.DefaultRouter()
router.register(r'groups', GroupViewSet)
```

```
class Group(models.Model):
  ...

  @authenticated_users
  def has_object_read_permission(self, request):
    return self.is_public or
      self.members.filter(pk=request.user.pk).exists()
```

```
class Group(models.Model):
  ...

  @authenticated_users
  def has_object_read_permission(self, request):
    return self.is_public or
      self.members.filter(pk=request.user.pk).exists()

  def has_object_write_permission(self, request):
    return self.membership_set \
      .get(user=request.user).is_admin
```

```
class Message(models.Model):
  group = models.ForeignKey(Group)
  author = models.ForeignKey("User")
  body = models.TextField()
```

```
class Message(models.Model):
  group = models.ForeignKey(Group)
  author = models.ForeignKey("User")
  body = models.TextField()

  def has_object_read_permission(self, request):
    return self.group.has_object_read_permission(request)
```

```
class Message(models.Model):
  group = models.ForeignKey(Group)
  author = models.ForeignKey("User")
  body = models.TextField()

  def has_object_read_permission(self, request):
    return self.group.has_object_read_permission(request)

  def has_object_write_permission(self, request):
    return self.author == request.user
```

```
class Message(models.Model):
  group = models.ForeignKey(Group)
  author = models.ForeignKey("User")
  body = models.TextField()

  def has_object_read_permission(self, request):
    return self.group.has_object_read_permission(request)

  def has_object_write_permission(self, request):
    return self.author == request.user

  def has_object_delete_permission(self, request):
    return self.author == request.user or
      self.group.has_object_write_permission(request)
```

```
class Comment(models.Model):
  message = models.ForeignKey(Message)
  commenter = models.ForeignKey("User")
  reply = models.TextField()
```

```
class Comment(models.Model):
  message = models.ForeignKey(Message)
  commenter = models.ForeignKey("User")
  reply = models.TextField()

  def has_object_read_permission(self, request):
    return self.message \
      .has_object_read_permission(request)
```

```
class Comment(models.Model):
  message = models.ForeignKey(Message)
  commenter = models.ForeignKey("User")
  reply = models.TextField()

  def has_object_read_permission(self, request):
    return self.message \
      .has_object_read_permission(request)

  def has_object_write_permission(self, request):
    return self.commenter == request.user
```

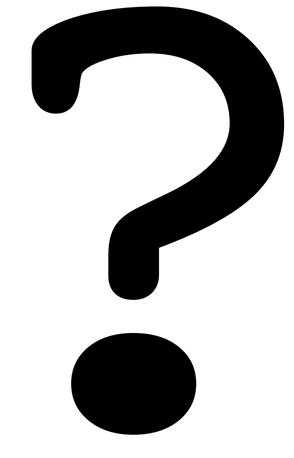
```
class Comment(models.Model):
  message = models.ForeignKey(Message)
  commenter = models.ForeignKey("User")
  reply = models.TextField()

  def has_object_read_permission(self, request):
    return self.message \
      .has_object_read_permission(request)

  def has_object_write_permission(self, request):
    return self.commenter == request.user

  def has_object_delete_permission(self, request):
    return self.commenter == request.user or
      self.message.has_object_delete_permission(request)
```

04

More

- Read the docs

Thank You



Presenter: Morten Barklund
@barklund

close