Django REST w praktyce

Marek Piechula
10-07-2017
v1

Spis treści

O mnie!

- Programuję komercyjnie od 5 lat
- Jestem **inżynierem** magistrem
- Lubię mikrokontrolery, astronomię i dobre mięsko

Trochę o **HTTP**, REST, CRUD

- Całkiem popularny protokół w internecie
- Server-Client: Klient żąda, Serwer odpowiada
- Żądanie posiada nagłówki, metodę, adres oraz ‘ciało’
- Odpowiedź posiada kod odpowiedzi, nagłówki oraz ‘ciało’

Trochę o HTTP, **REST**, CRUD

- REST **nie jest protokołem**, a *architekturą*
- Każdy zasób powinien mieć swój unikalny adres
- Każdy zasób powinien spełniać **CRUD**
- Serwis powinien być bezstanowy
- Serwis powinien nie być zależny od formatu danych - XML / JSON / HTML

Trochę o HTTP, REST, **CRUD**

CRUD	**HTTP**
Create	POST
Read	GET
Update	PUT
Partial **U**pdate	PATCH
Delete	DELETE

Czym jest Django Rest Framework

- Jest Frameworkiem w framework-u Django (Frameworkocepcja!)
- W fikuśny sposób obsługuje szeroko pojęty REST
- Bazuje na komponentach Django (jak ORM oraz Class Based Views)

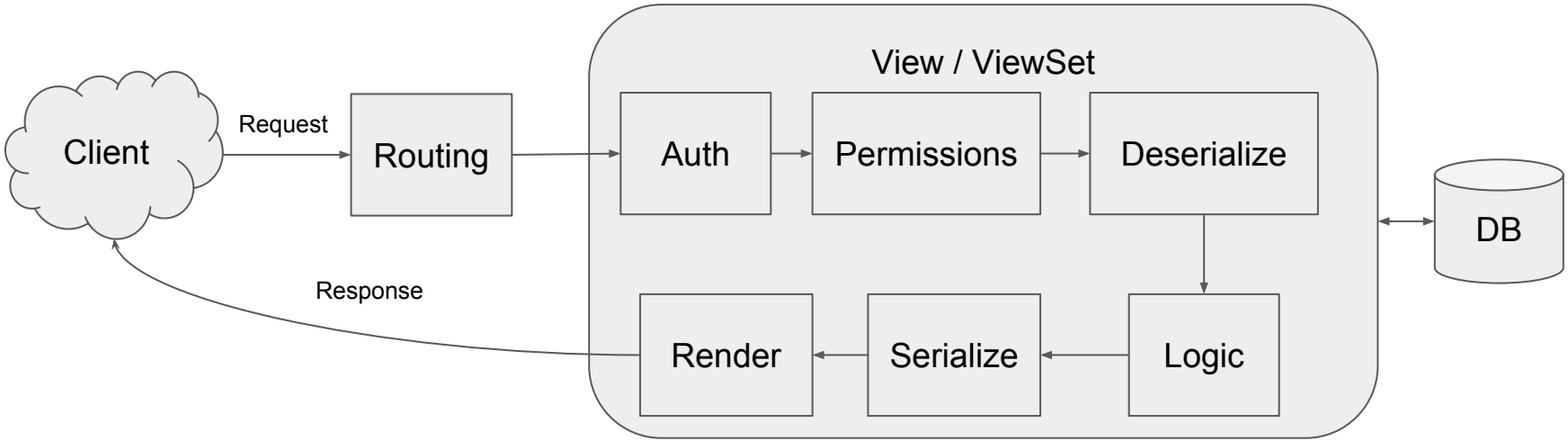
Zalety oraz wady DRF

- Bardzo dobrze integruje się z Django w każdym etapie projektu (początkowym lub ‘legacy’)
- Świetny samouczek oraz dokumentacja
- Bardzo czysty kod źródłowy
- Dość rozsądne rozbicie komponentów

Zalety oraz **wady** DRF

- *Django Way or Highway* - zrobienie inaczej niż DRF przewidział może boleć
- Wolny z powodu dużej ilość warstw oraz abstrakcji
- Pamięciożerny

Architekturalny podział DRF



Trudne widoki

1a. Chcemy mieć prosty widok tylko do odczytu

```
1.  class PagesViewSet(ReadOnlyModelViewSet):
2.      queryset = Page.objects.all()
3.      serializer_class = PageSerializer
4.
5.
6.  class PageSerializer(ModelSerializer):
7.
8.      class Meta:
9.          model = Page
10.         fields = ('id', 'title', 'user', 'description')
11.
12.
13.  router = SimpleRouter(trailing_slash=False)
14.  router.register(r'pages', views.PagesViewSet)
15.  urlpatterns = router.urls
```

Trudne widoki

1b. Chcemy mieć prosty widok tylko do odczytu tylko swoich stron

```
1.   class MyPagesViewSet(ReadOnlyModelViewSet):
2.      queryset = Page.objects.all()
3.      serializer_class = PageSerializer
4.      permission_classes = [IsAuthenticated]
5.
6.      def get_queryset(self):
7.          queryset =  super().get_queryset()
8.          if self.request.user.is_admin:
9.              return queryset
10.         return queryset.filter(user=self.request.user)
```

Trudne widoki

2. Chcemy wyświetlać i edytować swoje dane

```
class MyUserViewSet(UpdateModelMixin,
                    RetrieveModelMixin,
                    GenericViewSet):
    queryset = User.objects.all()
    permission_classes = [IsAuthenticated]
    serializer_class = MyUserSerializer

    def get_object(self, queryset=None):
        return self.request.user

class MyUserSerializer(ModelSerializer):

    class Meta:
        model = Page
        fields = ('id', 'email')
        read_only_fields = ('id',)
```

```
class DummyRouter(SimpleRouter):
   routes = [Route(
        url=r'^{prefix}$',
        mapping={'get': 'retrieve',
                 'put': 'update'},
        name='{basename}',
        detail=False,
        initkwargs={'suffix': 'Detail'},
    )]

drouter = DummyRouter()
drouter.register(
    r'my-user',
    views.MyUserViewSet)
urlpatterns = router.urls + drouter.urls
```

Trudne widoki

3. Dodatkowe akcje z notyfikacją po zapisaniu

```
1.   class NotificationViewSet(ModelViewSet):
2.       queryset = Notification.objects.all()
3.       serializer_class = NotificationSerializer
4.
5.       def perform_create(self, serializer):
6.           instance = serializer.save()
7.           self.send_emails(instance,  on='create')
8.
9.       def perform_update(self, serializer):
10.          instance = serializer.save()
11.          self.send_emails(instance,  on='update')
12.
13.      def send_emails(self, instance, on):
14.          print(f'HI {instance}!, on={on}')
```

```
1.   class NotificationSerializer(
2.           ModelSerializer):
3.
4.       class Meta:
5.           model = Notification
6.           fields = (
7.               'requester',
8.               'title',
9.               'invoke_on')
```

Trudne widoki

4. Wyświetlanie w szczegółach kategorii z jego przedmiotami (Zagnieżdżone obiekty)

```
1.   class CategoryViewSet(ModelViewSet):
2.       queryset = Category.objects.all()
3.
4.       def get_serializer_class(self):
5.           if self.action == 'list':
6.               return SimpleCategorySerializer
7.           return CategorySerializer
8.
9.
10.  class ItemSerializer(ModelSerializer):
11.
12.      class Meta:
13.          model = Item
14.          fields = ('name', 'value')
```

```
1.   class SimpleCategorySerializer(
2.           ModelSerializer):
3.
4.       class Meta:
5.           model = Category
6.           fields = ('id', 'name')
7.           read_only_fields = ('name',)
```

Trudne widoki

4. Wyświetlanie w szczegółach kategorii z jego przedmiotami

```
class CategorySerializer(ModelSerializer):
    items = ItemSerializer(many=True)

    class Meta:
        model = Category
        fields = ('id', 'name', 'items')

    def  create items(
            self, category, items_data):
        Item.objects.bulk create(
            Item(category=category, **data)
            for data in items_data
        )
```

```
    def create(self, validated data):
        items data = validated_data.pop(items')
        category = (
          Category.objects
          .create(**validated data))
        self._create_items(category, items_data)

        return category

    def update(self, instance, validated data):
        items data = validated data.pop(items')
        instance.items.all().delete()
        self. create items(instance, items_data)
        return super().update(
            instance, validated_data)
```

Trudne widoki

5. Dodatkowe wyliczone pola do modelu (np. Raport)

```
class UserReportViewSet(
        ReadOnlyModelViewSet):
    queryset = (
        User.objects
        .annotate(count_pages=Count('pages'))
        .all()
    )
    serializer class = UserReportSerializer1
    # serializer_class = UserReportSerializer2


class UserReportSerializer1(Serializer):
    id = IntegerField()
    email = CharField()
    is admin = BooleanField()
    is moderator = BooleanField()
    count_pages = IntegerField()
```

```
class UserReportSerializer1(ModelSerializer):

    class Meta:
        model = User
        fields = ('id', 'email', 'count_pages')
        read only fields = (
          'id', 'is_admin',
          'is_moderator', 'count_pages')

    def build unknown field(
            self, field name, model class):
        if field name == 'count pages':
            return IntegerField, {}
        super().build unknown field(
            field_name, model_class)

```

Trudne widoki

6. Tylko Admin lub właściciel może edytować swoje obiekty

```
1.  class RealPagesViewSet(ModelViewSet):
2.      queryset = Page.objects.all()
3.      serializer_class = PageSerializer
4.
5.      permission_classes = [IsAuthenticated, PagePermission]
6.
7.  class PagePermission(BasePermission):
8.      message = 'another user not allowed'
9.
10.     def has_object_permission( self, request, view, obj):
11.         if request.method == 'GET':
12.             return True
13.         user = request.user
14.         return user.is_admin or user.is_moderator or obj.user == request.user
```

Proste testy

```
1.  @pytest.mark.django_db
2.  def test_real_page_is_owner():
3.      page_user = create_user( 'b@b.pl')
4.      page = create_page(page_user,  'foo')
5.
6.      client = APIClient()
7.      client.force_login(page_user)
8.      response = client.patch(
9.          f'/api/real-pages/{page.id}',
10.         {'title': 'foobar'})
11.
12.     assert response.status_code ==  200
13.     assert response.data == {
14.         'id': page.id,
15.         'user': page_user.id,
16.         'title': 'foobar',
17.         'description': ''}
```

Protip

1. Nie nadużywaj z @action - to pogłębia logikę, którą jest bardzo trudno oddzielić jako osobny widok
2. Korzystaj z mixinów od DRF
3. Zamiast ID - UUID
4. Pamiętaj o paginacji jak kochasz swój RAM
5. Nie chowaj Logiki w Serializerach - od tego są ~~kontrolery~~ widoki
6. Etag + Last modified

Źródła

1. https://github.com/firemark/drf-best-practices
2. http://www.django-rest-framework.org/
3. https://github.com/encode/django-rest-framework
4. ~~https://www.filmweb.pl/film/Django-2012-620541~~
5. https://www.djangoproject.com/
6. https://github.com/jakubroztocil/httpie
7. https://www.jetbrains.com/pycharm/
8. https://docs.pytest.org/en/latest/

HACKERSPACE SILESIA

CODING DOJO SILESIA

Q/A

marpiechula@gmail.com

github.com/firemark/