Переопределенные классы APIView

ModelSerializer в сериализаторах

ModelSerializer — это уровень абстракции над сериализатором по умолчанию, который позволяет быстро создать сериализатор для модели в Django. Django REST Framework — это оболочка над Django Framework по умолчанию, которая в основном используется для создания различных API. Существует три этапа перед созданием API через инфраструктуру REST, преобразование данных модели в формат JSON/XML (сериализация), визуализация этих данных в представлении, создание URL-адреса для сопоставления с набором представлений.

Класс ModelSerializer предоставляет ярлык, позволяющий автоматически создавать класс Serializer с полями, соответствующими полям Model.

Класс ModelSerializer аналогичен обычному классу Serializer, за исключением того, что:

- Он автоматически сгенерирует набор полей на основе модели.
- Он автоматически сгенерирует валидаторы для сериализатора, такие как валидаторы unique_together.
- Он включает простые реализации по умолчанию .create()и .update().

```
class AccountSerializer(serializers.ModelSerializer):
    class Meta:
        model = Account
        fields = ['id', 'account_name', 'users', 'created']
```

```
class AccountSerializer(serializers.ModelSerializer):
    class Meta:
        model = Account
        # specify field names
        exclude = ['id']
```

```
class AccountSerializer(serializers.ModelSerializer):
    # defining fields manually
    url = serializers.CharField(source ='get_absolute_url', read_only = True)

    class Meta:
        # specify model
        model = Account
```

```
class AccountSerializer(serializers.ModelSerializer):
    class Meta:
        model = Account
        fields = ['id', 'account_name', 'users', 'created']
        # specify read only fields
        read_only_fields = ['account_name']
```

Обратите внимание, что внешний ключ обозначаем как в модели, а как в таблице, если у нас есть связи таблиц.

В DRF существует несколько предопределенных базовых классов

- CreateAPIView – создание данных по POST-запросу;
- ListAPIView – чтение списка данных по GET-запросу;
- RetrieveAPIView – чтение конкретных данных (записи) по GET-запросу;
- DestroyAPIView – удаление данных (записи) по DELETE-запросу;
- UpdateAPIView – изменение записи по PUT- или PATCH-запросу;
- ListCreateAPIView – для чтения (по GET-запросу) и создания списка данных (по POST-запросу);
- RetrieveUpdateAPIView – чтение и изменение отдельной записи (GET-, PUT- и PATCH-запросы);
- RetrieveDestroyAPIView – чтение (GET-запрос) и удаление (DELETE-запрос) отдельной записи;
- RetrieveUpdateDestroyAPIView – чтение, изменение и добавление отдельной записи (GET-, PUT-, PATCH- и DELETE-запросы).

Продолжим разработку энциклопедии. Теперь данные в сериализаторе можно изменять опираясь на модель.

```python
class MenSerializer(serializers.ModelSerializer):
    class Meta:
        model = Men
        fields = '__all__'
```

Перейдем к изменению представлении с помощью переопределенных классов.
Для возврата информации, используются Get запрос, который возвращает список данных. Для возврата списка с данными можно использовать 2 типа запросов: ListAPIView и ListCreateAPIView. Различие этих классов в том, что первый класс только вернет данные. А при вызове второго будет добавлена форма для добавления информации о новом человеке.

```python
from rest_framework.generics import ListCreateAPIView
class MenAPIView(ListCreateAPIView):
    queryset = Men.objects.all()
    serializer_class = MenSerializer
```


```
{
    "id": 2,
    "title": "Второй известный человек",
    "content": "Ну очень известный",
    "time_create": "2023-03-01T07:36:35.559104Z",
    "is_published": true,
    "cat": 2
},
{
    "id": 3,
    "title": "Третий известный человек",
    "content": "Ну очень известный",
    "time_create": "2023-03-01T07:51:22.769991Z",
    "is_published": true,
    "cat": 3
},
```


Можно переопределить атрибут queryset и задать критерии отбора показа информации о людях, добавленных за последние 2 дня (например).

```python
class MenAPIView(ListCreateAPIView):
    # queryset = Men.objects.all()
    serializer_class = MenSerializer

    def get_queryset(self):
        last_two_days = now() - timedelta(days=2)
        return Men.objects.filter(time_create__gt=last_two_days)
```

RetrieveAPIView, UpdateAPIView и DestroyAPIView Позволяют нам обращаться к одной из записей и определяя нужный метод для работы с данным

```python
from rest_framework.generics import ListCreateAPIView, RetrieveAPIView, DestroyAPIView, UpdateAPIView,
```

```python
class SingleMenView(RetrieveAPIView):
    queryset = Men.objects.all()
    serializer_class = MenSerializer
```

```python
class DelSingleMenView(DestroyAPIView):
    queryset = Men.objects.all()
    serializer_class = MenSerializer
```

```python
class UpSingleMenView(UpdateAPIView):
    queryset = Men.objects.all()
    serializer_class = MenSerializer
```

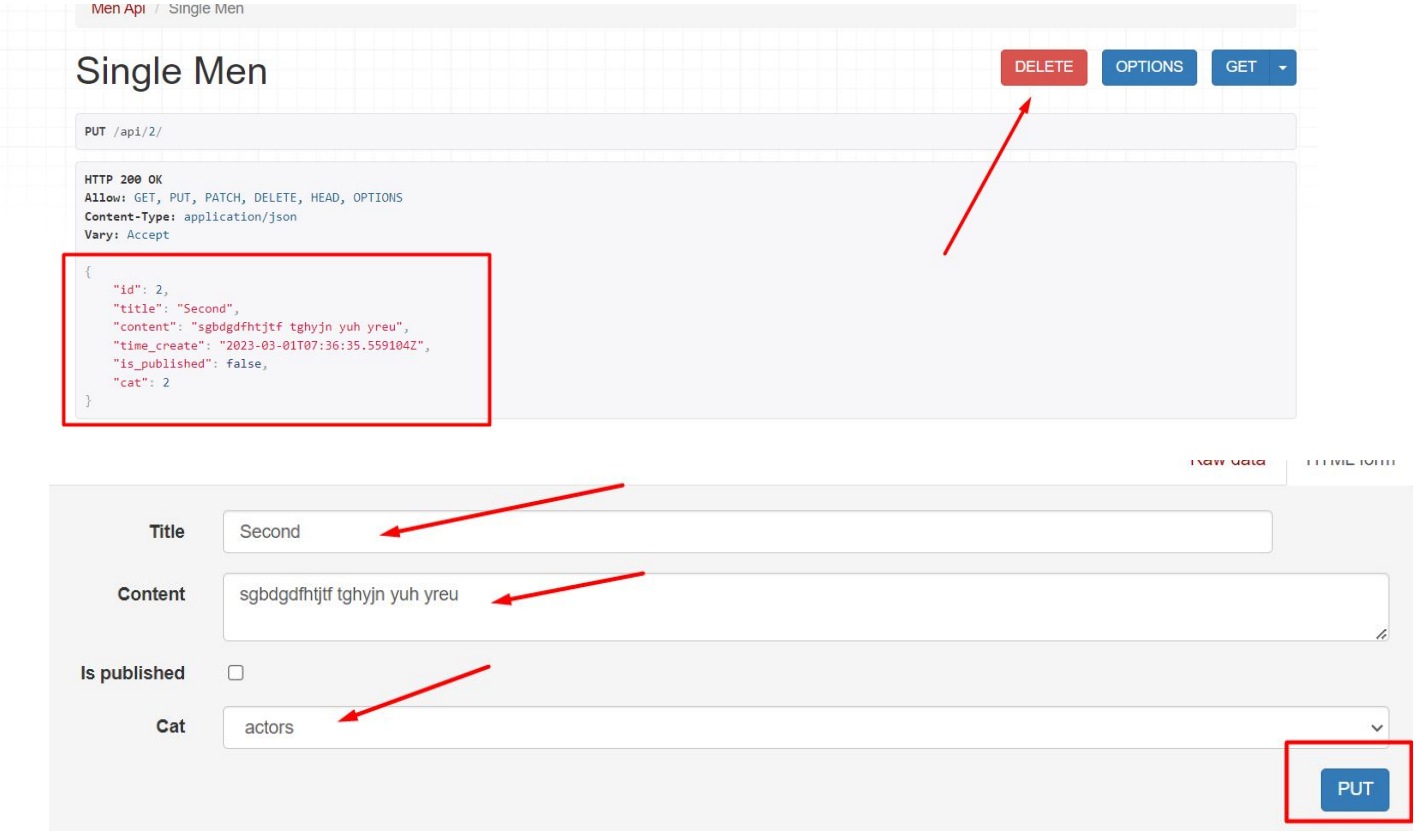
```python
urlpatterns = [
    path('', views.MenAPIView.as_view()),
    path('<int:pk>/', views.SingleMenView.as_view()),
    path('del/<int:pk>/', views.DelSingleMenView.as_view()),
    path('up/<int:pk>/', views.UpSingleMenView.as_view())
]
```

Но тогда придется написать разные маршруты для вызова этих методов на отдельных страницах в браузере. Что не очень удобно. Для решения Этой задачи можно использовать один класс RetrieveUpdateDestroyAPIView

```python
class SingleMenView(RetrieveUpdateDestroyAPIView):
    queryset = Men.objects.all()
    serializer_class = MenSerializer
```

```python
urlpatterns = [
    path('', views.MenAPIView.as_view()),
    path('<int:pk>/', views.SingleMenView.as_view()),
]
```

И тогда для проведения всех операции с одним элементом нам нужен всего один путь. Что и является правильной организацией узлов в Rest.
По результату мы при задании pk получим следующую страницу для работы с 1 записью из таблицы



Задание

Создать API для онлайн кинотеатра, используя классы ListAPIView и RetrieveUpdateDestroyAPIView.
Сериализатор можно создать с использованием ModelSerializers.
В наличии должны быть следующие таблицы:

- Фильмы: Название, Год выпуска, Страна, Режиссёр, Жанр
- Режиссер: ФИО, год рождения
- Жанр: Название жанра
- Афиша: Дата, Фильмы

Организовать связи один-ко-многим для таблиц Фильмы и Режиссер, Жанр; Афиша и Фильмы.
В API можно добавить, посмотреть, изменить и удалить информацию в любой таблице.