Лабораторная работа №5

Разработка API

Цель: научиться создавать полнофункциональный API в архитектурном стиле RESTful

Задачи:

1. Научиться создавать сериализаторы для моделей данных
2. Разработать RESTful API
3. Реализовать объекты ViewSet и маршрутизаторы

Ход работы

Разработка RESTful API

API в архитектурном стиле RESTful основаны на ресурсах. Модели представляют ресурсы, а HTTP-методы, такие как GET, POST, PUT, DELETE используются для извлечения, создания, обновления и удаления объектов. Коды HTTP-ответа также используются в этом контексте – возвращаются разные коды HTTP-ответа, чтобы указывать на результатах HTTP-запроса, например коды 2XX ответа относятся к успеху, 4XX относятся к ошибкам и т.д.

Наиболее распространенными форматами обмена данными RESTful API являются JSON и XML.

Разработаем RESTful API с сериализацией JSON к своему проекту. API будет обеспечивать следующую функциональность:

1. извлекать предметы
2. извлекать имеющиеся курсы
3. извлекать содержимое курса
4. зачислять на курс

API можно разрабатывать с нуля с помощью Django, создав конкретно-прикладные представления. При этом существуют несколько сторонних модулей, упрощающих создание API к проекту. Наиболее популярным из них является фреймворк DRF (Django REST framework).

Для разработки API мы будем использовать следующие компоненты:

- Сериализаторы
- Парсеры и рендеры
- Представления API
- URL-адреса

– Аутентификацию и разрешения.

1. Фреймворк DRF позволяет разрабатывать RESTful API для проектов. Установите его, выполнив команду:

```
py -m pip install djangorestframework= =3.15.1
```

2. Отредактируйте файл settings.py проекта educa, добавив rest_framework в настроечный параметр INSTALLED_APPS, чтобы активировать приложение:

```
INSTALLED_APPS = [
    #...
    'rest_framework',
]
```

3. Добавьте следующий код в файл settings.py:

```
REST_FRAMEWORK = {
    'DEFAULT_PERMISSION_CLASSES':[
'rest_framework.permissions.DjangoModelPermissionsOrAnonRead
Only'
    ]
}
```

Определение сериализаторов

После установки DRF необходимо задать способ сериализации данных. Выходные данные должны сериализироваться в конкретный формат, а выходные данные – десериализоваться для обработки.

Фреймворк DRF предоставляет следующие классы, чтобы формировать сериализаторы для одиночных объектов:

Serialoizer – обеспечивает сериализацию обычных экземпляров класса Python

ModelSerializer – обеспечивает сериализацию экземпляров модели

HyperlinkedModelSerializer – тот же, что ModelSerializer, но представляет объектные зависимости со ссылками, а не с первичными ключами.

1. Создадим первый сериализатор. Внутри каталога приложения courses создайте следующую файловую структуру:

```
api/
    __init__.py
serializers.py
```

Всю функциональность API будем создавать внутри каталога api.

2. Отредактируйте файл settings.py добавив следующий код:

```
from rest_framework import serializers
```

```
from courses.models import Subject
class SubjectSerializer (serializers.ModelSerializer):
    class Meta:
        model=Subject
        fields=['id','title','slug']
```

Это сериализатор для модели Subject. Сериализаторы определяются аналогично классам Django Form и ModelForm. Meta-класс позволяет указывать подлежащую сериализации модель и поля, которые нужно включать в сериализуцию. Если поле fields не задавать, то в сериализацию будут включены все модели.

Разработка представлений списка и детальной информации

Фреймворк DRF поставляется с набором обобщенных представлений и примесных классов, которые можно использовать для разрботки API-представлений.

1. Создадим представления списка и детальной информации, чтобы извлекать объекты Subject. Внутри каталога courses/api/ создайте новый файл и назовите его views.py. Добавьте в него следующий код:

```
from rest_framework import generics
from courses.api.serializers import SubjectSerializer
from courses.models import Subject

class SubjectListView(generics.ListAPIView):
    queryset = Subject.objects.all()
    serializer_class = SubjectSerializer

class SubjectDetailView(generics.RetrieveAPIView):
    queryset = Subject.objects.all()
```

В данном коде используются обобщенные представления ListAPIView, RetrieveAPIView фреймворка DRF. URL-параметр pk представлен в представлении детальной информации, чтобы извлекать объект по заданному первичному ключу.

2. Добавим шаблон url-адресов представлений. Внутри каталога courses/api/ создайте новый файл, назовите его urls.py и придайте ему следующий вид:

```
from django.urls import path
from . import views
app_name = 'courses'
```

```
urlpatterns=[
    path( 'subjects/',
            views.SubjectListView.as_view(),
            name='subject_list'
    ),
    path('subjects/<pk>/',
    views.SubjectDetailView.as_view(),
    name = 'subject_detail'
    ),
]
```

В шаблон URL-адреса для представления SubjectDetailView вставлен параметр pk URL-адреса, чтобы получить объект с заданным первичным ключом модели Subject, которым является поле id. Отредактируйте главный файл urls.py проекта educa, вставив шаблон URL-адреса API:

```
 urlpatterns = [
#...
path('api/',include('courses.api.urls',namespace='api'))
,
 ]
```

Именное пространство api используется для URL-адресов API. Начальные конечные точки API теперь готовы к использованию.

Потребление API

1. Проверим работу API. Запустите сервер разработки и перейдите по адресу http://127.0.0.12:8000/api/subjects/ в своем браузере, вы увидите доступный для просмотра API-интерфейс фреймворка DRF - страницу списка предметов в доступном для просмотра API-интерфейсе фреймворка DRF.

Этот html-интерфейс предоставляется рендером BrowableAPIRender и отображает результирующие заголовки и содержимое, позволяет выполнить запросы.

2. Пройдите по URL-адресу http://127.0.0.1:8000/api/subjcts/1 в совем браузере. Вы увидите, что один объект Subject будет транслирован в формат JSON.

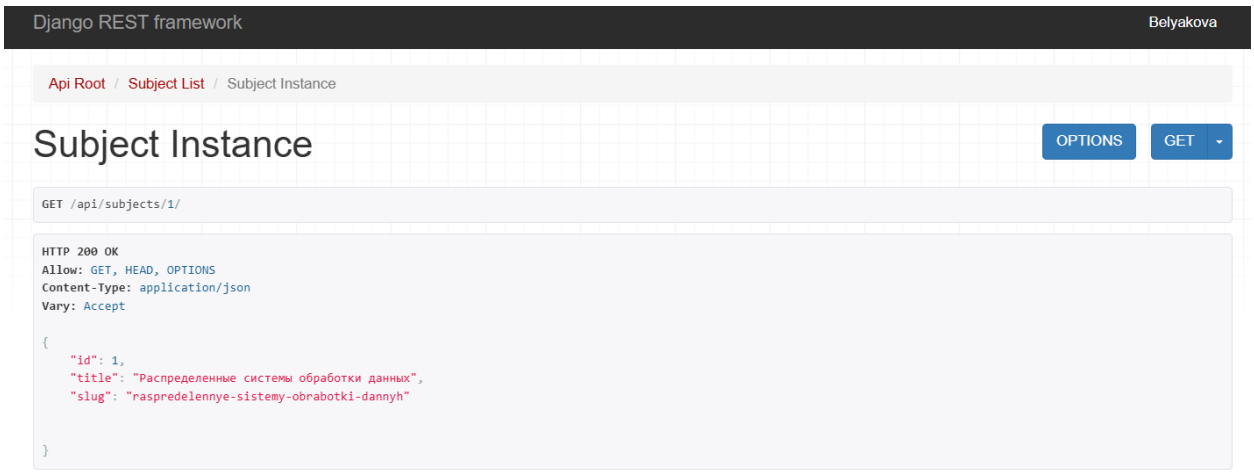

Рисунок 1 – Страница детальной информации о предмете в доступном для просмотра API-интерфейсе фреймворка DRF.

Расширение сериализаторов

Добавление дополнительных полей в сериализаторы

Отредактируем представления предметов, чтобы включить в них количество имеющихся курсов по каждому предмету. Для этого будем использовать агрегатные функции Django, чтобы аннотировать количество связанных курсов по каждому предмету.

1. Отредактируйте файл api/views.py приложения courses, добавив следующий код:

```
from django.db.models import Count
class SubjectListView(generics.ListAPIView):
    queryset =
Subject.objects.annotate(total_courses=Count('courses'))
    serializer_class = SubjectSerializer

class SubjectDetailView(generics.RetrieveAPIView):
    queryset =
Subject.objects.annotate(total_courses=Count('courses'))
    serializer_class=SubjectSerializer
```

Теперь для представлений SubjectListView и SubjectDetailView будем использовать объект QuerySet. Представления SubjectListView и SubjectDetailView применяют функцию агрегирования Count для аннотирования количества связанных курсов.

2. Отредактируем файл api/serializers.py приложения courses, добавив код, выделенный жирным шрифтом:

```
class SubjectSerializer (serializers.ModelSerializer):
```

```
    total_courses=serializers.IntegerField()
    class Meta:
        model=Subject
        fields=['id','title','slug','total_courses']
```

Поле total_courses будет получать свое значение автоматически из атрибута total_courses сериализуемого объекта.

Пройдите по URL-адресу http://127.0.0.1:8000/api/subjects/1/ в своем браузере. Сериализованный объект JSON теперь содержит атрибут total_courses.

Реализация полей методов сериализатора

Фреймворк DRF содержит класс SerializerMethodField, позволяющий реализовать поля, доступные только для чтения, которые получают свое значение путем вызова метода класса-сериализатора.

1. Создадим метод, который сериализует три самых популярных курса по предмету. Предметы будем ранжировать по количеству записанных на них студентов. Отредактируйте файл api/serializers.py приложения courses, добавив в него код, выделенный жирным.

```
    from django.db.models import Count
    class SubjectSerializer (serializers.ModelSerializer):
        total_courses=serializers.IntegerField()
        popular_courses = serializers.SerializerMethodField()

        def get_popular_courses(self, obj):
            courses = obj.courses.annotate(total_students=Count('students')).order_by('total_students')[:3]
            return [
                f'{c.title} ({c.total_students})' for c in courses
            ]


        class Meta:
            model=Subject

fields=['id','title','slug','total_courses','popular_courses']
```

Пройдите по url-адресу http://127.0.0.1:8000/api/subjects/1/ в своем браузере и вы увидите, что сериализованный объект JSON теперь содержит атрибут total_courses.

Добавление постраничной разбивки в представления

Фреймворк DRF содержит встроенные возможность постраничной разбивки, чтобы управлять количеством объектов, передаваемых в API. Когда содержимое сайта начнет расти, предметов и курсов будет много и понадобиться постраничная разбивка.

Давайте обновим представление SubjectListView, чтобы включить постраничную разбивку. Создайте новый файл в каталоге courses/api/ и назовите его pagination.py. Добавьте в него следующий код:

```
from rest_framework.pagination import PageNumberPagination

class StandardPagination(PageNumberPagination):
    page_size=10
    page_size_query_param ='page_size'
    max_page_size=50
```

Этот класс наследует от класса PageNumberPagination, который обеспечивает поддержку постраничной разбивки на основе номеров страниц.

Отредактируйте файл api/views.py приложения courses, добавив следующие строки, выделенные жирным шрифтом:

```
from courses.api.pagination import StandardPagination
class SubjectListView(generics.ListAPIView):
    queryset                                                    =
Subject.objects.annotate(total_courses=Count('courses'))
    serializer_class = SubjectSerializer
    pagination_class=StandardPagination
```

Пройдите по URL-адресу http://127.0.0.1:8000/api/subjects/ в своем браузере. Вы увидите, что структура JSON теперь отличается из-за постраничной разбивки.

Пройдите по URL-адресу http://127.0.0.1:8000/api/subjects/?page_size=2&page=1 в своем браузере. Результаты будут разбиты на страницы по два элемента на страницу и выведена первая страница результатов.

Таким образом, мы создали конечные точки API для представлений предметов.

Разработка сериализатора курса

Создадим сериализатор для модели Course.

1. Отредактируйте файл api/serializers.py приложения courses, добавив код, выделенный жирным шрифтом:

```
from courses.models import Course, Subject
class CourseSerializer(serializers.ModelSerializer):
    class Meta:
        model=Course
        fields=[
            'id',
            'subject',
            'title',
            'slug',
            'overview',
            'created',
            'owner',
            'modules',
        ]
```

Сериализатор объекта Course готов к использованию.

Сериализация отношений

Фреймворк DRF поставляется с разными типами связанных полей, чтобы предосталять взаимосвязи между моделями.

Отредактируйте файл api/serializers.py приложения courses, добавив коды, выделенный жирным шрифтом:

```
class CourseSerializer(serializers.ModelSerializer):
    modules=serializers.StringRelatedFiels(many=True,
read_only=True)
        class Meta:
         #...
```

Поле modules обеспечивает сериализацию связанных объектов Module.

Создание вложенных сериализаторов

Если необходимо включать больше информации о каждом модуле, то необходимо сериализовать объекты Module и вкладывать их друг в друга.

1. Отредактируйте файл api/serializers.py приложения courses, добавив коды, выделенный жирным шрифтом:

```
from courses.models import Course, Module, Subject
```

```
class ModuleSerializer(serializers.ModelSerializer):
    class Meta:
        model=Module
        fields=['order','title','description']

class CourseSerializer(serializers.ModelSerializer):
    modules=ModuleSerializer(many=True, read_only=True)
    class Meta:

        model=Course

        fields=[
            'id',
            'subject',
            'title',
            'slug',
            'overview',
            'created',
            'owner',
            'modules',
        ]
```

Класс ModuleSerializer обеспечивает сериализацию для модели Module. Затем меняется атрибут modules в классе CourseDerializer, чтобы вложить сериализатор ModuleSerializer.

Создание наборов представлений и маршрутизаторов

Объекты ViewSet(наборы представлений) позволяют определять взаимодействия API и дают возможность фреймворку DRF возможность создавать URL-адреса с помощью объекта Router. Использование ViewSet позволяет избегать повторения логики в нескольких представлениях. Объекты ViewSet содержат действия для следующих стандартных операций: create(), list(), retrieve(), update(), destroy().

1. Создадим объект ViewSet для модели Course. Отредактируйте файл api/views.py добавив в него код, выделенный жирным шрифтом:

```
from rest_framework import viewsets
from courses.api.serializers import CourseSerializer, SubjectSerializer
from courses.models import Course, Subject
```

```
class CourseViewSet(viewsets.ReadOnlyModelViewSet):
    queryset = Course.objects.prefetch_related('modules')
    serializer_class = CourseSerializer
    pagination_class = StandardPagination
```

Класс CourseViewSet наследует от ReadOnlyModelViewSet, который предлагает действия list(), retrieve() только для чтения, чтобы выводить список объектов или получать один объект.

2. Отредактируйте файл api/urls.py, создав маршрутизатор для объекта ViewSet:

```
from django.urls import include, path
from rest_framework import routers
from . import views
app_name = 'courses'
router=routers.DefaultRouter()
router.register('courses', views.CourseViewSet)
urlpatterns=[
   path('',include(router.urls)),
]
```

Пройдите по URL-адресу http://127.0.0.1:8000/api/ в своем браузере. Вы увидите, что маршрутизатор выводит список всех объектов ViewSet по своему URL-адресу.

1. Конвертируем представления SubjectListView, SubjectDetailView в один объект ViewSet. Отредактируйте файл api/views.py, удалив либо закомментировав классы SubjectListView и SubjectDetailView. Добавьте следующий код:

```
class CourseViewSet(viewsets.ReadOnlyModelViewSet):
    queryset = Course.objects.prefetch_related('modules')
    serializer_class = CourseSerializer
    pagination_class = StandardPagination
```

2. Отредактируйте файл api/urls.py и удалите следующие URL-адреса, так как они больше не нужны:

```
from django.urls import include, path
from rest_framework import routers
from . import views

app_name = 'courses'

router=routers.DefaultRouter()
router.register('courses', views.CourseViewSet)
```

```
router.register('subjects', views.SubjectViewSet)

urlpatterns=[
#    path( 'subjects/',
#             views.SubjectListView.as_view(),
#             name='subject_list'
#    ),
#    path('subjects/<pk>/',
#    views.SubjectDetailView.as_view(),
#    name = 'subject_detail'
#    ),
    path('',include(router.urls)),
    #path('courses/<pk>/enroll/',
views.CourseEnrollView.as_view(), name='course_enroll'),

]
```

Пройдите по URL-адресу http://127.0.0.1:8000/api/ в своем браузере, вы увидите, что маршрутизатор теперь содержит URL-адреса для наборов представлений ViewSets courses и subjects.

Обобщенные API-представления и объекты ViewSet очень полезны для создания Rest API на основе собственных моделей и сериализаторов. Однако возникает необходимость реализовывать собственные представления с конкретно-прикладной логикой.

Контрольные вопросы

1. Какую роль выполняют сериализаторы в DRF?
2. В чем разница между Serializer и ModelSerializer? Какой вы использовали и почему?
3. Покажите и объясните код вашего сериализатора.
4. Как с помощью сериализатора можно включать связанные объекты