Лабораторная работа №6

Разработка конкретно-прикладных API представлений

Цель работы: изучение возможностей использования класса APIView для формирования функциональности API сверху функциональности класса View

Задачи:

1. Изучить особенности использования класса ViewSet и APIView
2. Реализовать аутентификацию с использованием ViewSet
3. Реализовать внешние API функции

Ход работы

1. Создадим представление зачисления пользователей на курсы. Для этого отредактируем файл api/views.py и добавим код, выделенный жирным шрифтом:

```
from django.shortcuts import get_object_or_404
from rest_framework.response import Response
from rest_framework.views import APIView
#...
class CourseEnrollView(APIView):
    def post(self, request, pk, format=None):
        course = get_object_or_404(Course, pk=pk)
        course.students.add(request.user)
        return Response({'enrolled': True})
```

Данный код создает конкретно-прикладное представление как подкласс APIView. Для действий POST определяется метод post(). Никакой другой http-метод в этом представлении не разрешен. Ожидаемый параметр pk содержит идентификатор курса. По нему извлекается курс, иначе возвращается ошибка 404. Текущий пользователь добавляется во взаимосвязь students «многие ко многим» объекта Course и возвращается успешный ответ.

2. Отредактируйте файл api/urls.py добавив следующий url-адреса представления CourseEnrollView:

```
path(   'courses/<pk>/enroll/',
 views.CourseEnrollView.as_view(),
name='course_enroll'
),
```

Теперь можно выполнять запрос методом Post для зачисления студентов на курс, но также необходима возможность идентифицировать пользователя и предотвращать доступ неавторизованных пользователей.

Разработка аутентификации. Реализация базовой аутентификации

Аутентификацию можно задавать для каждого представления либо установить ее глобально для параметра DEFAULT_AUTHENTICATION_CLASSES.

Отредактируйте файл api/views.py приложения courses, добавив атрибут authentication_classes в представление CourseEnrollView:

```
from rest_framework.authentication import BasicAuthentication
#...
class CourseEnrollView(APIView):
    authentication_classes=[BasicAuthentication]
```

Пользователи будут идентифицироваться по учетным данным, установленным в заголовке Authorization HTTP-запроса.

Добавление разрешений в представление

Фреймворк DRF содержит систему разрешений, служащую для ограничения доступа к представлениям. Если пользователю отказано в представлении, то он получает http-код ошибки 401 (не авторизован) или 403 (отказано в доступе).

1. Отредактируйте файл api/views.py приложения courses, добавив атрибут permission_classes в класс CourseEnrollView:

```
from rest_framework.permissions import IsAuthenticated
#...
class CourseEnrollView(APIView):
    authentication_classes=[BasicAuthentication]
    permission_classes=[IsAuthenticated]
```

Здесь указано разрешение IsAuthenticated, оно будет предотвращать доступ анонимных пользователей к представлению. Теперь можно выполнять запрос методом POST к новому методу API.

Добавление дополнительных действий в объекты ViewSet

В объекты ViewSet можно добавлять дополнительные действия. Поменяем ранее созданное представление CourseEnrollView на конкретно-прикладное действие объекта ViewSet.

1. Отредактируйте файл api/views.py видоизменив класс CourseViewSet:

```
from rest_framework.decorators import action
class CourseViewSet(viewsets.ReadOnlyModelViewSet):
    queryset = Course.objects.prefetch_related('modules')
    serializer_class = CourseSerializer

    @action(
        detail=True,
        methods=['post'],
        authentication_classes=[BasicAuthentication],
        permission_classes=[IsAuthenticated]
    )
    def enroll(self, request, *args, **kwargs):
        course = self.get_object()
        course.students.add(register.user)
        return Response({'enrolled':True})
```

В этом исходном коде добавляется конкретно-прикладной метод enroll(), который представляет дополнительное действие этого объекта ViewSet.

2. Отредактируйте файл api/urls.py, удалив лиюо закомментировав класс CourseEnrollView.

URL-адрес зачисления на курсы теперь автоматически генерируется маршрутизаторы. Остается прежним, так как он создается динамически с использованием имени enroll-действия.

Создание конкретно-прикладных решений

Для того, чтобы студенты имели доступ только к содержимому тех курсов, на которые они были записаны применим конкретно-прикладной класс разрешений.

Внутри каталога courses/api/ создайте новый файл и назовите его premissions.py и добавьте в него следующий код:

```
from rest_framework.permissions import BasePermission

class IsEnrolled(BasePermission):
    def has_object_permission(self, request, view, obj):
```

```
            return
obj.students.filter(id=request.user.id).exists()
```

Здесь BasePermission определяется как подкласс и переопределяется метод has_object_permission(). Выполняется проверка на то, что пользователь, который делает запрос присутствует во взаимосвязи students объекта Course.

Сериализация содержимого курса

1. Отредактируйте файл api/serializers.py приложения courses, добавив в него следующий код:

```
from courses.models import Content, Course, Module, Subject
class ItemRelatedField(serializers.RelatedField):
    def to_representation(self, value):
        return value.render()

class ContentSerializer(serializers.ModelSerializer):
    item=ItemRelatedField(read_only=True)
    class Meta:
        model=Content
        fields=['order','item']
```

В этом коде определяется конкретно-прикладное поле путем подклассирования поставляемого с фреймворком DRF поля-сериализатора RelatedField и переопределения метода to_representation(). Для модели Content определяется сериализатор ContentSerializer и для обобщенного внешнего ключа item используется конкретно-прикладное поле.

2. Создадим альтернативный сериализатор для модели Module, включающий его содержимое, а также расширенный сериализатор Course.

Для этого отредактируйте файл api/serializers.py и добавьте в него код:

```
class ModuleWithContentSerializer(
    serializers.ModelSerializer):
    contents = ContentSerializer(many=True)
    class Meta:
        model=Module
        fields= ['order','title','description','contents']

class CourseWithContentsSerializer(
    serializers.ModelSerializer):
    modules = ModuleWithContentSerializer(many=True)
    class Meta:
        model=Course
```

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

3. Создадим представление, которое имитирует поведение действия retrieve(), но включает содержимое курса. Для этого отредактируйте файл api/views.py, добавьте в него следующий код:

```
from courses.api.permissions import IsEnrolled
from courses.api.serializers import CourseWithContentsSerializer

class CourseViewSet(viewsets.ReadOnlyModelViewSet):
#...
@action(
        detail=True,
        methods=['get'],
        serializer_class=CourseWithContentsSerializer,
        authentication_classes=[BasicAuthentication],
        permission_classes=[IsAuthenticated, IsEnrolled]
    )
    def contents(self, request, *args, **kwargs):
        return self.retrieve(request, *args, **kwargs)
```

Пройдите по url-адресу http://127.0.0.1:8000/api/courses/1/contents и вы увидите, что каждый модуль включает в себя прорисованный html содержимого курса.

Таким образом, мы создали простой API, который позволяет другим сервисам программно обращаться к приложению courses. Фреймворк DRF также позволяет создавать и редактировать объекты с помощью класса ModelViewSet.

Потребление RESTful API

С API можно взаимодействовать, используя интерфейс JavaScript Fetch API во фронтэнде нашего приложения. Создадим простое приложение,

которое использует RESTful API для извлечения всех имеющихся курсов, а затем зачисляет студента на все эти курсы.

1. Откройте командную оболочку и установите библиотеку requests:

```
py –m pip install requests= =2.31.0
```

2. Рядом с каталогом educa создайте новый каталог и назовите его api_examples. Внутри каталога api_examples/ создайте новый файл и назовите его enroll_all.py. У вас должна получиться файловая структура следующего вида:

```
api_examples/
     enroll_all
educa/
     …
```

3. Отредактируйте код файла enroll_all.py, добавив в него следующий код:

```
import requests

base_url = 'http://127.0.0.1:8000/api/'
url = f'{base_url}courses/'
available_courses = []

while url is not None:
    print(f'Loading courses from {url}')
    r = requests.get(url)
    response = r.json()
    url = response['next']
    courses=response['results']
    available_courses+=[course['title'] for course in courses]
print(f'Available courses: {",".join(available_courses)}')
```

4. Запустите сервер разработки из каталога проекта educa

5. В другой командной оболочке выполните следующую команду из каталога api_examples

```
py enroll_all.py
```

Вы увидите результат со списком всех названий курсов.

Это ваш первый автоматический вызов API.

6. Отредактируйте файл enroll_all.py так, чтобы он выглядел следующим образом:

```
import requests
```

```
username=''
password=''

base_url = 'http://127.0.0.1:8000/api/'
url = f'{base_url}courses/'
available_courses = []

while url is not None:
    print(f'Loading courses from {url}')
    r = requests.get(url)
    response = r.json()
    url = response['next']
    courses=response['results']
    available_courses+=[course['title'] for course in
courses]
print(f'Available courses: {",".join(available_courses)}')

for course in courses:
    course_id=course['id']
    course_title = course['title']
    r = requests.post(
        f'{base_url}courses/{course_id}/enroll/',
        auth=(username,password)
    )
    if r.status_code==200:
       print(f'Successfully enrolled in {course_title}')
```

7. Значения переменный username и password замените учетными данными известного вам пользователя.

8. В командной оболочки выполните следующую команду из каталога api_examples/:

```
py enroll_all.py
```

В результате вы успешно зачислите пользователя на все имеющиеся курсы с помощью API.

Контрольные вопросы

1. Объясните, что такое REST и перечислите его ключевые архитектурные ограничения (принципы). Как Django REST Framework (DRF) помогает следовать этим принципам при разработке API?

2. Какова основная роль сериализаторов в DRF?

3. Чем отличается использование APIView от ViewSet? В каких сценариях предпочтительнее использовать ModelViewSet?

4. Назовите основные классы аутентификации и разрешений, доступные в DRF из коробки. Как можно создать собственный класс разрешений, например, чтобы разрешить доступ только автору объекта?

5. Для чего нужна пагинация в API? Какие стандартные классы пагинации предоставляет DRF? Как организовать фильтрацию списка объектов, например, по полю course_id?