Django Starter

Django и REST, ч.2. Реализация простого API с DRF

Introduction

Лазорык Михаил

Software developer, 3 года опыта

mykhailo.lazoryk

mykhailo-lazoryk

Django и REST, ч.2. Реализация простого API с DRF

План урока

1. Представление в RESTful API: FunctionBasedView, ClassBasedView, ViewSets, Generic views
2. Сериализация и валидация
3. Роутинг
4. Аутентификация и авторизация
5. Система прав доступа

Представление в RESTful API. ClassBasedView

REST framework предоставляет класс APIView, который является подклассом джанговских классов View. Классы APIView имеют следующие отличия от обычных классов View:

- Запросы, переданные обработчику, будут экземплярами *Request REST framework*, а не джанговских *HttpRequest*.
- Обрабатывающие методы могут возвращать *Response REST framework* вместо джанговских *HttpResponse*. Представление будет осуществлять согласование содержимого и устанавливать нужный рендерер для ответа.
- Любые исключения APIException будут выявлены и связаны с соответствующими ответами.
- Входящие запросы будут подтверждены и соответствующее разрешение и/или проверки будут произведены перед тем, как передать запрос на обработку.

Использование класса APIView не особо отличается от использования обычных классов View. Как правило, входящий запрос отправляется на обработку соответствующему методу, как .get() или .post(). Для класса, который отвечает за различные аспекты поведения API, могут задаваться дополнительные атрибуты.

Представление в RESTful API. FunctionBasedView

REST framework позволяет работать с обычным представлениями на основе функций. В его составе есть набор простых декораторов, которые оборачивают ваши представления-функции для того, чтобы обеспечить получение экземпляра Request (вместо обычного джанговского HttpRequest) и позволяют возвращать Response (вместо HttpResponse), а также позволяют настроить каким образом обрабатывается запрос.

@api_view():

- представление по умолчанию использует классы рендера, парсера, аутентификации и т.д., которые прописаны в настройках.

- по умолчанию используются только методы GET. Другие методы вызовут сообщение «405 Method Not Allowed». Для того, чтобы изменить это, необходимо указать методы в представлении.

Для того, чтобы переписать настройки по умолчанию, в REST framework есть набор дополнительных декораторов, которые можно добавить к вашим представлениям. Они должны прописываться после декоратора @api_view. Например, чтобы создать представление, которое с помощью тротлинга (от англ. throttling это механизм (функция) защиты процессора от перегрева) делает так, что определенный пользователь может вызвать представление только один раз в день. Можно воспользоваться декоратором @throttle_classes, который передает список классов тротлинга.

Представление в RESTful API. ViewSets

- Django REST framework позволяет комбинировать логику для набора связанных представлений в одном классе, который называется ViewSet. В других фреймворках вы также можете встретить похожие концепции под названием «Resources» или «Controllers».

- Класс ViewSet это просто класс-представление, которое не использует никаких методов обработки, как .get() или .post(), а вместо этого включает действия .list() и .create().

- Обработчики метода для ViewSet связаны только для соответствующих действий на моменте окончательной обработке представления, используя метод .as_view().

- Как правило, вместо того, чтобы подробно регистрировать представления в viewset в urlconf, вы регистрируете viewset в классе маршрутизатора, который автоматически определяет для вас urlconf.

Представление в RESTful API. ViewSets

Класс **ViewSet** дает два главных преимущества перед классом **View**:

- Можно заключить неоднократно повторяющуюся логику в один класс. Потребуется лишь один раз уточнить queryset, и после этого он будет использоваться во множестве представлений.

- Используя маршрутизаторы нам больше не нужно самим писать **URL conf**. Однако эти плюсы несут свои компромиссы. Использование обычных представлений и URL confs более очевидно и предоставляет больше контроля. **ViewSets** полезны если вы хотите, чтобы все заработало как можно быстрее, или когда у вас большой **API**, и вам требуется обеспечить равномерную конфигурацию **URL** во всем проекте.

Представление в RESTful API. ViewSets

- Средства REST framework предоставляют маршрутизаторы для стандартных операций create/retrieve/update/destroy.
- Класс ViewSet наследуется от APIView. Вы можете использовать любой из стандартных атрибутов, такие как permission_classes, authentication_classes, чтобы контролировать поведение API в viewset.
- Класс ViewSet не реализует действия. Для того, чтобы воспользоваться классом ViewSet, нужно переписать класс и расписать действия.
- Класс GenericViewSet наследуется от GenericAPIView и предоставляет стандартный набор методов get_object, get_queryset и другие общие механизмы поведения представления, но при этом не реализует их.
- Для того, чтобы использовать GenericViewSet, вам нужно переписать класс и, либо создать миксины требуемых классов, либо явно определить реализацию действий.
- Класс ModelViewSet наследуется от GenericAPIView и реализует различные действия, совмещая функционал различных классов миксинов.
- Класс ModelViewSet предоставляет следующие действия .list(), .retrieve(), .create(), .update(), .partial_update(), и .destroy().

Представление в RESTful API. Generic views

- GenericAPIView - этот класс расширяет класс APIView, реализуя часто повторяющееся поведение. Каждое общее представление строится путем комбинации GenericAPIView с одним из классов-миксинов.

- Одно из ключевых преимуществ представлений-классов заключается в том, что они позволяют использовать повторяющиеся паттерны. REST framework реализует эту идею через встроенные представления. Общие представления REST framework позволяют быстро строить представления API, которые тесно связаны с вашими моделями баз данных. Если общие представления не подходят целям вашего API, вы всегда можете отказаться от них в пользу обычных классов APIView или повторно использовать миксины и базовые классы, используемые в общих представлениях для того, чтобы создать свой набор многократно используемых общих представлений.

- Как правило, при использовании общих представлений вы должны переписать ваше представление и установить несколько атрибутов класса. Для более сложных классов вам также может понадобиться переписать различные методы класса представления. Для самых простых случаев вам может понадобиться передать любой атрибут класса с помощью метода .as_view().

Представление в RESTful API. Generic views

CreateAPIView:
Используется для создающих конечных точек.
Предоставляет: обработчик метода post.
Расширяет: GenericAPIView, CreateModelMixin

ListAPIView
Используется для создания неизменяемых конечных точек для набора экземпляров модели.
Предоставляет: обработчик метода get.
Расширяет: GenericAPIView, ListModelMixin

RetrieveAPIView
Используется для создания неизменяемых конечных точек для экземпляра одной модели.
Предоставляет: обработчик метода get.
Расширяет: GenericAPIView, RetrieveModelMixin

DestroyAPIView
Используется для создания только удаляемых конечных точек для экземпляра одной модели.
Предоставляет: обработчик метода delete.
Расширяет: GenericAPIView, DestroyModelMixin

UpdateAPIView
Используется для создания только дополняемых конечных точек для экземпляра одной модели.
Предоставляет: обработчик методов put и patch.
Расширяет: GenericAPIView, UpdateModelMixin

Представление в RESTful API. Generic views

ListCreateAPIView

Используется для конечных точек считывания и записи для набора экземпляров модели.
Предоставляет: обработчик методов get и post.
Расширяет: GenericAPIView, ListModelMixin, CreateModelMixin

RetrieveUpdateAPIView

Используется для чтения и дополнения конечных точек для экземпляра одной модели.
Предоставляет: обработчик методов get, put и patch.
Расширяет: GenericAPIView, RetrieveModelMixin, UpdateModelMixin

RetrieveDestroyAPIView

Используется для чтения или удаления конечных точек для экземпляра одной модели.
Предоставляет: обработчик методов get и delete.
Расширяет: GenericAPIView, RetrieveModelMixin, DestroyModelMixin

RetrieveUpdateDestroyAPIView

Используется для чтения-записи-удаления конечных точек для экземпляра одной модели.
Предоставляет: обработчик методов get, put, patch и delete.
Расширяет: GenericAPIView, RetrieveModelMixin, UpdateModelMixin, DestroyModelMixin

Сериализация и валидация

- Сериализаторы в REST framework работают аналогично классам Django Form и ModelForm. Мы предоставляем класс Serializer, который дает вам мощный, общий способ управления вашими ответами, а также класс ModelSerializer - полезный и быстрый способ создания сериализаторов, которые имеют дело с экземплярами модели и querysets.

- Объявление сериализатора очень похоже на объявление формы.

- Если мы хотим иметь возможность возвращать полные экземпляры объектов на основе проверенных данных, нам нужно реализовать один или оба метода .create() и update(). При десериализации данных вам всегда нужно вызвать is_valid(), прежде чем пытаться получить доступ к проверенным данным или сохранить экземпляр объекта. Если возникнут какие-либо ошибки проверки, свойство .errors будет содержать словарь, представляющий сообщения об ошибках.

- В отдельные поля в сериализаторе можно включить валидаторы, объявив их в экземпляре поля.

- Классы сериализаторов могут также включать повторно используемые валидаторы, которые применяются к полному набору данных поля. Эти валидаторы подключаются путем объявления их во внутреннем мета-классе.

Роутинг

Некоторые веб фреймворки, как **Rails**, автоматически реализуют механизм логической связи URL'ов приложения с входящими запросами. **REST framework** добавляет поддержку автоматического роутинга для Джанго, тем самым предоставляя пользователю простой и надежный способ написания логики представлении для набора URL.

Ниже приводится пример простого URL conf с использованием **SimpleRouter:**

```
from rest_framework import routers

router = routers.SimpleRouter()
router.register(r'users', UserViewSet)
router.register(r'accounts', AccountViewSet)

urlpatterns = router.urls
```

Роутинг

Метод **register**() должен включать два обязательных аргумента:

- prefix - префикс URL, использующийся с данным набором роутеров.
- viewset - класс viewset.

Опционально вы можете указать дополнительный аргумент:

- **base_name** - основа для использования с URL именами. Елси аргумент не указан, то базовое имя будет автоматически сгенерировано на основе атрибута **queryset** из **viewset**, при наличии такого. Обратите внимание, что если **viewset** не включает атрибут **queryset**, то вы должны использовать **base_name** при регистрации **viewset**.

Пример выше генерирует следующие **URL** паттерны:

- URL pattern: ^users/$ Name: “user-list”
- URL pattern: ^users/{pk}/$ Name: “user-detail”
- URL pattern: ^accounts/$ Name: “account-list”
- URL pattern: ^accounts/{pk}/$ Name: “account-detail”

Аутентификация и авторизация

Аутентификация - это механизм связывания входящего запроса с набором идентифицирующих учетных данных, таких как пользователь, от которого поступил запрос, или токен, с которым он был залогинен. Затем permission и throttling могут использовать эти учетные данные, чтобы определить, следует ли разрешить запрос.

Платформа REST предоставляет ряд схем аутентификации «из коробки», а также позволяет реализовывать пользовательские схемы.

Можно установить глобальную схему авторизации:
REST_FRAMEWORK = {“DEFAULT_**AUTHENTICATION**_CLASSES”: [
“rest_framework.authentication.**BasicAuthentication**”,
“rest_framework.authentication.**SessionAuthentication**”,
]}

Также можно использовать для каждого представления свою схему авторизации.

Аутентификация и авторизация

Permissions (**доступа**) - отвечает за то, получит реквест доступ, или нет.

Они запускаются перед всеми процессами.

Есть следующие доступы:

- AllowAny
- IsAuthenticated
- IsAdminUser
- IsAuthenticatedOrReadOnly
- DjangoModelPermissions
- DjangoModelPermissionsOrAnonReadOnly
- DjangoObjectPermissions
- Custom permissions

План урока

1. Представление в RESTful API: FunctionBasedView, ClassBasedView, ViewSets, Generic views
2. Сериализация и валидация
3. Роутинг
4. Аутентификация и авторизация
5. Система прав доступа

Проверка знаний

TestProvider.com

TestProvider – это online сервис проверки знаний по информационным технологиям. С его помощью Вы можете оценить Ваш уровень и выявить слабые места. Он будет полезен как в процессе изучения технологии, так и для общей оценки знаний IT специалиста.

Проверьте как Вы усвоили данный материал на TestProvider.com

Успешное прохождение финального тестирования позволит Вам получить соответствующий Сертификат.

Спасибо за внимание! До новых встреч!

Лазорык Михаил
Software developer

tieto

django

Информационный видеосервис для разработчиков программного обеспечения

ITVDN

IT VIDEO DEVELOPERS NETWORK

TestProvider PROMETRIC TEST CENTER CyberBionic systematics Microsoft Partner Silver Learning Windows Azure Cloud Partner EBA European Business Association