GeekBrains

Django REST Framework

Введение в REST и Django REST Framework

[python>= 3]

На этом уроке

1. Узнаем, для чего используется REST.
2. Рассмотрим назначение каждого из видов REST-запросов.
3. Научимся подключить Django REST Framework (DRF) к django-проекту.
4. Создадим простое API для одной модели данных.

Оглавление

О курсе

Содержание

Приветствуем вас на курсе Django REST Framework. В нём мы рассмотрим наиболее актуальные технологии разработки Веб-приложений:

- строгое разделение приложения на front-end и back-end;
- создание API на стороне back-end и использование для этих целей DRF и GraphQl;
- создание front-end с использованием React;
- сборка проекта в Docker.

Основную часть внимания уделим back-end и DRF, так как наша специализация — глубокое изучение Python. Но также мы рассмотрим основные моменты и принципы при разработке front-end-части на React и настройке взаимодействия между front-end и back-end.

Практическое задание (проект)

Во время выполнения задания мы будем разрабатывать индивидуальный проект «Веб-сайт для работы с TODO-заметками». Задания составлены таким образом, что позволяют:

1. Применить и закрепить рассмотренный на занятии материал.
2. Создать следующую часть проекта.
3. Понять определённую логику и последовательность разработки проекта с самого начала.

На выходе мы получим современный проект, построенный с учётом актуальных технологий и принципов программирования.

Введение в REST

Основы REST

Representational State Transfer (REST) — архитектурный стиль разработки ПО, который в основном используется при создании веб-сервисов. Он включает в себя набор ограничений и правил. Рассмотрим основные ограничения и правила.

Единообразие интерфейса

Каждой единице информации соответствует определённый **ресурс** (url-адрес), по которому эту информацию можно получить. Например, по адресу /animals/ можно получить список животных, по адресу /animals/1/ — информацию об одном животном и т. д.

Для ресурса доступен определённый набор операций — **глаголов**. В более узком смысле — это тип запроса, отправляемого на данные url.

В зависимости от типа запроса, отправленного на один и тот же адрес, можно совершать разные действия. REST расширяет стандартные типы запросов GET и POST и добавляет новые. Например, PUT, PATCH, DELETE и другие. Каждый тип подробно рассмотрим ниже. Зная тип запроса и адрес, мы чётко понимаем, что должно произойти, если имеем дело с REST.

Клиент — Сервер

Все данные и бизнес-логика хранятся на сервере, который предоставляет клиенту стандартизированный интерфейс для доступа к этим данным.

Отсутствие состояния

Сервер не хранит в себе информацию о состоянии клиента. Теперь пользователь сам заботится о хранении состояния. Например, оно может храниться в кэше браузера или каким-то другим способом.

Есть и другие ограничения. Подробнее о них и о развитии и возникновении REST можно узнать, перейдя по ссылкам в дополнительных материалах.

Важно! *Для разработки API с помощью REST не существует единого стандарта. REST — это архитектурный стиль, а не протокол, хотя иногда можно услышать выражение REST-протокол. Обычно разработчики следуют принятым в сообществе стандартам, например, используют HTTP, JSON.*

Разработка веб-приложений с помощью REST

HTTP

Большинство разработчиков для создания веб-приложений используют HTTP протокол. Вероятно, вам уже знакомы 2 основных типа запросов — GET и POST. Но есть ещё и другие, например, PUT, DELETE. Они позволяют расширить варианты работы с HTTP и в полной мере задействовать архитектуру REST.

При совместном использовании HTTP и REST url-адрес, на который мы отправляем запрос — это ресурс. Тип запроса определяет действие — что необходимо сделать с этим ресурсом.

Основные типы запросов и назначение каждого из них

- GET — чтение данных;
- POST — создание ресурса;
- PUT — создание или полная замена ресурса;
- DELETE — удаление ресурса;
- OPTIONS — описание параметров соединения с целевым ресурсом;
- HEAD — чтение заголовков запросов (аналогично GET, но без тела);
- PATCH — частичная модификация ресурса.

Таким образом, чтобы посмотреть, какие методы доступны по адресу /animals/, можно отправить OPTIONS-запрос. Для просмотра списка животных — GET /animals/. Создать новое животное — POST /animals/. Посмотреть одно животное — GET /animals/1. Для изменения животного — PUT /animals/1. Удалить животное — DELETE /animals/1. Если нам нужно просто проверить, доступен ли адрес, можем отправить на него HEAD-запрос.

JSON и API

В веб-приложении, во время обработки сервером запроса, Server Side Rendering (далее SSR) возвращал готовую html-страницу пользователю в виде текста, а Клиент (браузер) её отрисовывал.

Чтобы сделать Клиент-Сервер независимым и трансформировать работу с сервером в более гибкую, нужен другой способ.

1. Необходимо отделить сами данные от их представления. Мы будем представлять данные в стандартизированном виде. Для этого подходит формат JSON или аналогичные популярные форматы (например, XML, YAML…).
2. Необходим Application Programming Interface (далее API) для доступа к этим данным и возможности совершать с ними определённые действия.

Итоговый стек технологий

Подводя итог вышесказанному, для создания современного гибкого и расширяемого веб-сервиса нам понадобится:

- REST API;
- HTTP — протокол, включая большинство доступных запросов: GET, POST, PUT, DELETE, OPTIONS;
- JSON — для представления данных в стандартизированном виде.

К счастью, нам не требуется реализовывать все эти концепции с самого начала. Мы можем воспользоваться Django REST Framework (далее DRF) — для быстрого создания API внутри Django-проекта. С DRF мы можем создать веб-сервис с REST API либо комбинировать SSR и API.

Введение в Django REST Framework

О библиотеке

DRF — популярная библиотека для быстрой и удобной разработки REST API на основе django-фреймворка. Библиотека — фреймворк для разработки API. Она имеет собственный сайт, на котором можно найти полную документацию и небольшой Tutorial по её использованию.

В документации подробно описаны все возможности фреймворка. Наша задача — научиться отделять главное от второстепенного и применять технологию для разработки веб-приложений. Нужно научиться пользоваться документацией, чтобы понимать, где можно посмотреть узко-специфицированную информацию.

Установка и настройка проекта

Чтобы начать использовать DRF, создадим django-проект.

```
django-admin startproject <имя проекта>
```

Работа с django-фреймворком остаётся прежней, DRF расширяет, а не изменяет Django.

Сам DRF устанавливается как сторонняя библиотека.

```
pip install djangorestframework
```

Рекомендуется сразу установить markdown для удобной работы с API в браузере.

```
pip install markdown
```

И django-filter. Библиотека поможет добавлять удобные фильтры для данных как в django, так и в drf.

```
pip install django-filter
```

Далее добавляем rest_framework в список приложений settings.py-проекта.

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

Подключаем в urls.py-проекта адреса для авторизации DRF.

```
urlpatterns = [
    ...
    path('api-auth/', include('rest_framework.urls'))
]
```

На этом установка DRF закончена. В будущем в settings.py мы будем добавлять новые глобальные настройки для DRF и пользоваться классами и методами, доступными в библиотеке.

Создание REST API для простой модели данных

Модели

Как и в django, основой нашего приложения будут служить модели данных, связанные с базой данных. В большинстве случаев именно для этих моделей мы будем создавать REST API.

Кроме моделей, DRF вводит понятия — сериализаторы, рендеры, парсеры, роутеры и другие. Подробнее каждую из этих частей мы рассмотрим на следующих занятия. А пока разберём самый базовый случай создания API.

Это модель, не связанная с внешними ключами с другими моделями.

Для демонстрации создадим проект «Библиотека».

```
django-admin startproject library
```

Перейдём в папку с проектом и создадим приложение authors.

```
python manage.py startapp authors
```

В models.py создадим модель, описывающую автора, у которого есть имя, фамилия и год рождения.

```
from django.db import models


class Author(models.Model):
   first_name = models.CharField(max_length=64)
   last_name = models.CharField(max_length=64)
   birthday_year = models.PositiveIntegerField()
```

/authors/models.py

Подключим наше приложение в settings.py в INSTALLED_APPS.

Выполняем миграции.

```
python manage.py makemigrations
python manage.py migrate
```

Сохраним несколько авторов в базу данных. Это можно сделать любым удобным способом, например, через стандартную админку или скриптом или через shell.

Далее установим DRF и подключаем его к проекту.

```
pip install djangorestframework
pip install markdown       .
pip install django-filter
```

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

/library/settings.py

```
urlpatterns = [
    ...
    path('api-auth/', include('rest_framework.urls'))
]
```

/library/urls.py

Сериализаторы (Serializers)

После того как у нас появились модель и данные, создадим для них REST API. DRF позволяет удобно и быстро создать API сразу для всех методов GET, PUT, POST, DELETE.

Наше API будет предоставлять пользователю следующие возможности:

- смотреть список авторов;
- создавать нового автора;
- смотреть одного автора;
- изменять одного автора;
- удалять автора.

Для просмотра данных нам необходимо преобразовать модель Author в json. Поэтому DRF добавляет новый слой сериализаторов (Serializers).

Создадим простой сериализатор для модели Authors.

Важно! *При работе с проектом рекомендуется придерживаться определённой структуры. Это нужно, чтобы мы сами и другие разработчики в команде могли быстро работать с кодом. Правила не такие жёсткие, как в django, и обычно устанавливаются командой проекта. Нам же рекомендуется придерживаться правил из этой методички и примеров кода. Например, Serializers мы будем хранить в специальном модуле проекта serializers.py.*

Создадим новый модуль serializers.py в нашем приложении.

В нём напишем следующий код сериализатора.

```
from rest_framework.serializers import HyperlinkedModelSerializer
from .models import Author


class AuthorModelSerializer(HyperlinkedModelSerializer):
   class Meta:
       model = Author
       fields = '__all__'
```

/authors/serializers.py

Обратите внимание, Serializer чем-то похож на Django Form. Он может быть связан или нет с моделью. В нём есть набор полей для отображения.

Рассмотрим код подробнее:

```
class AuthorModelSerializer(HyperlinkedModelSerializer):
```

В примере мы используем сериализатор, связанный с моделью Author. Это значит, что сериализатор на основании неё будет создавать JSON-представление.

```
fields = '__all__'
```

Мы указываем, какие поля из модели будут отображаться в json. __all__ означает все поля.

Наборы представлений (View Sets)

Слой представлений (view) в нашем проекте сохраняется. Дополнительно DRF предоставляет возможность группировать несколько представлений в набор (это удобно). Чтобы создать набор представлений для модели Author в файле views.py, напишем следующий код:

```
from rest_framework.viewsets import ModelViewSet
from .models import Author
from .serializers import AuthorModelSerializer


class AuthorModelViewSet(ModelViewSet):
   queryset = Author.objects.all()
   serializer_class = AuthorModelSerializer
```

/authors/views.py

Рассмотрим код поподробнее:

```
class AuthorModelViewSet(ModelViewSet):
```

Мы используем наследование от `ModelViewSet`. Это означает, что набор views связан с моделью и будет работать с её данными.

```
queryset = Author.objects.all()
```

`queryset` указывает, какие данные мы будем выводить в списке.

```
serializer_class = AuthorModelSerializer
```

`serializer_class` определяет тот `Serializer`, который мы будем использовать.

viewset определяет, какие данные будут вводиться, а serializer назначает их представление в JSON.

Важно! *Каждую часть нашего приложения помещаем в свой модуль. Так как наш проект будет содержать только API, поместим ViewSet в файл views. Если на сайте будут API и классические views с SSR, то лучше их разместить в разных файлах.*

Routers. Urls

И последняя часть нашего простого приложения — это адреса (urls). На них пользователь будет отправлять запросы для работы с данными. В отличие от классического сайта на Django с SSR, теперь на адрес может поступать не 1–2, а сразу несколько различных запросов. Поэтому для их

формирования удобно использовать Router — специальное расширение (класс) DRF для формирования сразу нескольких адресов, начинающихся в одной точке.

В корневом файле urls.py напишем следующий код:

```
from django.contrib import admin
from django.urls import path, include
from rest_framework.routers import DefaultRouter
from authors.views import AuthorModelViewSet

router = DefaultRouter()
router.register('authors', AuthorModelViewSet)

urlpatterns = [
   path('admin/', admin.site.urls),
   path('api-auth/', include('rest_framework.urls')),
   path('api/', include(router.urls)),
]
```

/library/urls.py

Разберём код по частям:

```
router = DefaultRouter()
router.register('authors', AuthorModelViewSet)
```

Создаём объект класса DefaultRouter и связываем `AuthorModelViewSet` с адресом `authors`.

```
path('api/', include(router.urls)),
```

Подключаем адреса (urls), которые формирует роутер, к нашему проекту.

Результат

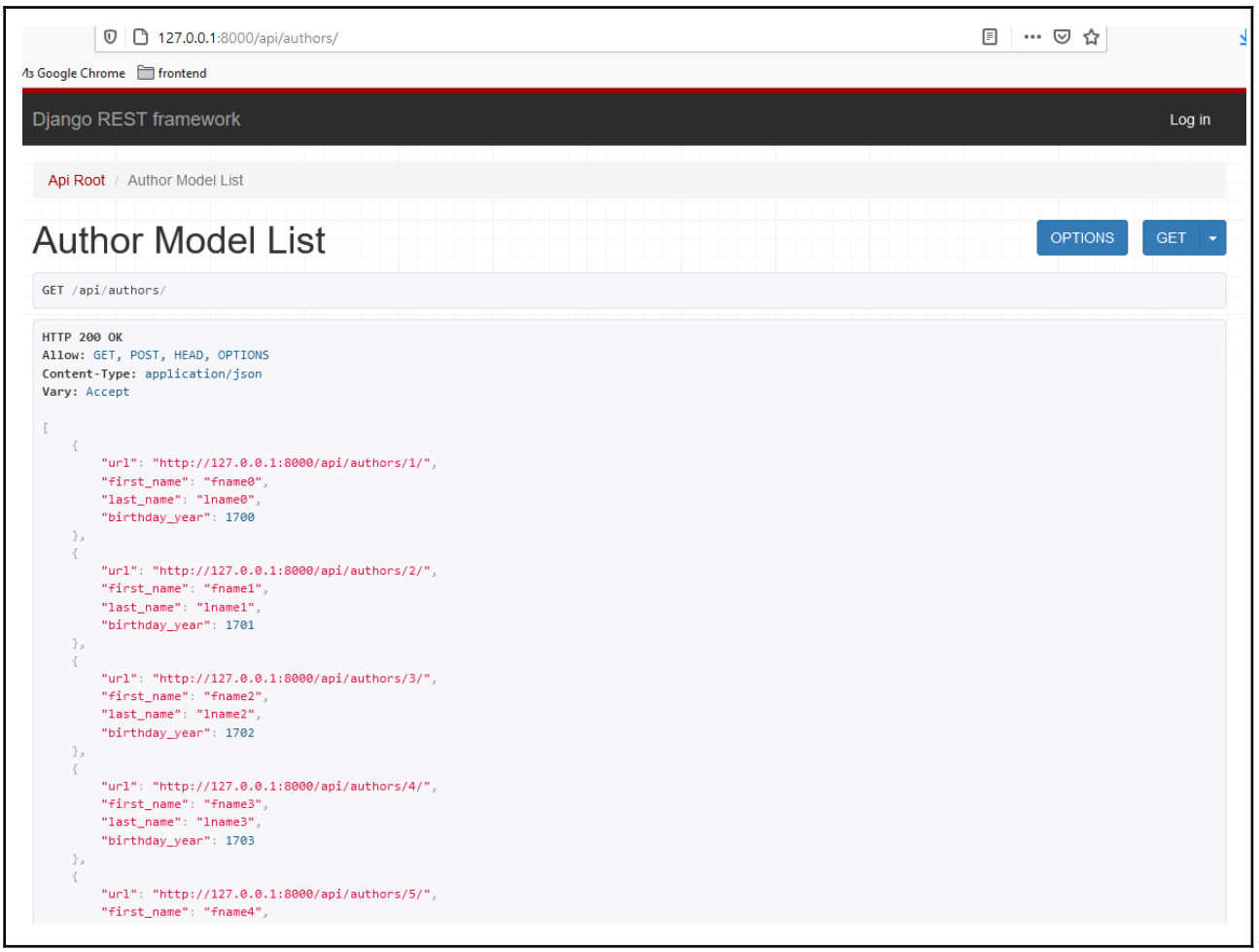
Проверим, что получилось. Запускаем тестовый сервер:

```
python mange.py runserver
```

И переходим по адресу /api/.

Примерно такой результат мы должны увидеть в браузере:

Мы видим, все доступные нашему API адреса, сейчас — это authors. Далее, если перейти по ссылке /api/authors/, увидим список авторов в формате json.



Важно! *Такой удобный интерфейс для работы с API в браузере нам предоставляет DRF. Если получить эти данные другим способом, например, с помощью библиотеки requests, получим просто текст в формате json, без всякой html-разметки и кнопок управления.*

Обратите внимание на кнопки с названиями запросов GET, POST, OPTIONS: они позволяют отправить на один адрес разные типы запросов и получить соответствующие результаты.

Если перейти по адресу одного автора, например, /authors/1, то нам будут доступны запросы PUT и DELETE.

Таким образом, мы небольшими усилиями, используя несколько строк кода и несколько новых специальных классов из DRF, создали полное API для работы с моделью Authors.

Предлагаем самостоятельно попробовать все доступные типы запросов и посмотреть на результат их работы.

Итоги

На занятии мы рассмотрели основы REST API и создали API для простой модели данных. Выделили новые основные части DRF и посмотрели, как они используются в базовом виде. На следующих занятиях рассмотрим более сложные варианты использования API, типовые задачи (авторизация, права и т. д.), а также узнаем, как взаимодействовать с нашим API с помощью React.

Глоссарий

1. **API (Application Programming Interface)** — программный интерфейс приложения.
2. **DRF (Django REST Framework)** — библиотека для создания REST API с помощью Django-фреймворка.
3. **GraphQl** — язык запросов, альтернатива REST.
4. **React** — библиотека для создания пользовательских интерфейсов на JavaScript.
5. **REST (Representational State Transfer)** — архитектурный стиль разработки ПО.
6. **Router** — часть DRF, отвечающая за создание url-адресов.
7. **Serializer** — часть DRF, отвечающая за преобразование данных в формат json и обратно.
8. **View Set** — часть DRF, набор представлений для группировки нескольких представлений.

Дополнительные материалы

1. Официальный сайт Django REST Framework.
2. Расширение стандартной модели пользователя Django.
3. Стандартная админка Django.
4. Custom Management Commands.
5. Wiki REST.
6. HTTP Methods.

Используемые источники

1. Официальная документация DRF.
2. REST простым языком.
3. REST и HTTP.
4. Методы HTTP и их назначение.

Практическое задание

На протяжении всего курса в практическом задании мы по частям будем разрабатывать проект «Web-сервис для работы с TODO-заметками». Вот примерное описание работы всего сайта.

На сайте будет 3 категории пользователей: администраторы, менеджеры проектов и разработчики. Сайт позволяет работать с TODO-заметками, которые относятся к какому-либо проекту. Можно добавить на сайт проект и затем сохранять по нему заметки, содержащие текст и статус. Одним проектом занимается команда разработчиков и менеджеров.

После каждого занятия:

- сразу будем применять полученные знания на практике;
- добавлять в проект новые опции, относящийся к теме занятия.

В первом практическом задании создадим пользователя и API для работы с ним.

В самостоятельной работе отработаем умения

- подключать DRF к проекту;
- создавать REST API для простой модели данных.

Зачем?

Для использования базовых классов DRF в дальнейших проектах.

Последовательность действий

1. Создать новый проект на github или gitlab.
2. Создать django-проект.
3. Установить DRF и подключить его к django-проекту.
4. Создать приложение для работы с пользователем.
5. Создать свою модель пользователя.

6. В ней поле email сделать уникальным.
7. Сделать для неё базовое API — по аналогии модели Author. В качестве полей выбрать username, firstname, lastname, email. Если выбрать все поля, при попытке сериализации может возникнуть ошибка сериализации связанного поля. Эту тему мы рассмотрим далее.
8. Подключить стандартную админку.
9. Создать суперпользователя.
10. (Задание со *) Создать management command — скрипт для запуска через manage.py для автоматического создания суперпользователя и нескольких тестовых пользователей (Management commands).
11. Сдать работу в виде ссылки на репозиторий с кодом.