GeekBrains

Django REST Framework

Авторизация. Система прав

На этом уроке

1. Рассмотрим виды авторизации.
2. Узнаем о видах прав.
3. Научимся добавлять базовую авторизацию, авторизацию по токену.
4. Разберёмся, как использовать права групп.

Оглавление

Система прав

Введение

Практически каждый проект содержит пользователей, которые обладают разными правами. Например, гость может смотреть одни страницы, зарегистрированный пользователь — другие, и так далее.

Так как REST API не имеет состояния, и пользователь не хранится в сессии, система прав и авторизации в DRF отличается от механизма в Django.

DRF содержит собственный механизм прав и авторизации. В этом занятии мы рассмотрим основные возможности работы с правами и авторизацией. Затем реализуем наиболее подходящий вариант для демонстрационного примера.

Работа с правами

Механизм выдачи прав хорошо описан в [соответствующем разделе официальной документации](#). Рассмотрим принципы его работы.

Настройка прав для всего проекта

Обычно в settings.py указываются права по умолчанию для всего проекта, например:

```
REST_FRAMEWORK = {
    'DEFAULT_PERMISSION_CLASSES': [
        'rest_framework.permissions.IsAuthenticated',
    ]
}
```

/settings.py

В этом примере rest_framework.permissions.IsAuthenticated говорит, что все запросы доступны только авторизованным пользователям. DRF содержит различные виды прав, которые рассмотрим ниже. Права, указанные в settings.py, автоматически применяются ко всем ресурсам, если не будут переопределены локально.

Локальное указание прав

Чтобы изменить права для одной View или ViewSet, укажем их локально, например:

```
from rest_framework.permissions import IsAuthenticated
from rest_framework.views import APIView

class ExampleView(APIView):
    permission_classes = [AllowAny]
...
```

/<appname>/views.py

Свойство permission_classes, заданное локально, переопределяет права для этого View. Например, здесь ExampleView будет доступно для любого пользователя и гостя (AllowAny).

Расширение прав по умолчанию

При необходимости можно создавать собственные права, путём расширения имеющихся, например:

```
from rest_framework.permissions import BasePermission

class StaffOnly(BasePermission):
    def has_permission(self, request, view):
        return request.user.is_staff
```

В этом примере мы создали новое право наследованием от BasePermisson. Метод has_permission возвращает либо True, либо False. Здесь правами будут наделены пользователи-сотрудники (is_staff).

Виды прав

Рассмотрим основные виды прав, встроенных в DRF:

1. AllowAny — доступ есть у всех пользователей, включая неавторизованных.
2. IsAuthenticated — доступ есть только у авторизованных пользователей.
3. IsAdminUser — доступ есть только у администратора.
4. IsAuthenticatedOrReadOnly — доступ есть у авторизованных пользователей, у неавторизованных — доступ только на просмотр данных.
5. DjangoModelPermissions — использует систему прав Django на модели. Для каждой модели у пользователя могут быть права add, change, delete, view.
6. DjangoModelPermissionsOrReadOnly — аналогично DjangoModelPermissions, но с правом на просмотр у пользователей, не обладающих другими правами.

Можно создавать собственные права и использовать сторонние библиотеки.

Выбор прав и работа с ними

Настройка демонстрационного проекта

В следующих уроках в качестве демонстрационного проекта поработаем с Library. В прошлом занятии мы создали front-end-часть с маршрутизацией на стороне клиента. Сейчас мы добавим back-end-часть, авторизацию и права. А в следующем занятии объединим back-end и front-end.

Создадим django-проект:

```
django-admin startproject library
```

terminal

Установим DRF:

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

terminal

Добавим rest_framework в INSTALLED_APPS:

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

/library/settings.py

Создадим приложение mainapp^

```
python manage.py startapp mainapp
```

/terminal

В файле с моделями напишем следующий код:

```
from django.db import models


class Author(models.Model):
    name = models.CharField(max_length=64, unique=True)
```

```
    birthday_year = models.PositiveIntegerField()

    def __str__(self):
        return self.name


class Book(models.Model):
    name = models.CharField(max_length=64, unique=True)
    author = models.ForeignKey(Author, on_delete=models.CASCADE)

    def __str__(self):
        return self.name
```

/mainapp/models.py

Это модель автора и книги, для них создадим REST API.

Создадим в приложении файл serializers.py со следующим кодом:

```
from rest_framework import serializers
from .models import Author, Book

class AuthorSerializer(serializers.ModelSerializer):
    class Meta:
        model = Author
        fields = '__all__'


class BookSerializer(serializers.ModelSerializer):
    class Meta:
        model = Book
        fields = '__all__'
```

/mainapp/serializers.py

Далее во views.py напишем код View sets:

```
from rest_framework import viewsets
from .models import Author, Book
from .serializers import AuthorSerializer, BookSerializer


class AuthorViewSet(viewsets.ModelViewSet):
    serializer_class = AuthorSerializer
    queryset = Author.objects.all()


class BookViewSet(viewsets.ModelViewSet):
    serializer_class = BookSerializer
```

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

/mainapp/view.py

И в файле urls.py проекта подключим urls с помощью роутера:

```
from django.contrib import admin
from django.urls import path, include
from rest_framework import routers
from rest_framework.authtoken import views
from mainapp.views import AuthorViewSet, BookViewSet

router = routers.DefaultRouter()
router.register('authors', AuthorViewSet)
router.register('books', BookViewSet)

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

/library/urls.py

Сделаем миграции и запустим сервер.

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

/terminal

REST API для моделей готово. Теперь можно настроить нужные нам права.

Выбор и настройка прав

Выдадим права следующим образом:

1. Администратор имеет доступ ко всему.
2. Младшие сотрудники имеют доступ к созданию, удалению, изменению модели Book. Всё остальное могут только просматривать.
3. Старшие сотрудники имеют доступ к созданию, удалению, изменения моделей Author и Book. Остальное могут только просматривать.
4. Все остальные имеют права только на просмотр.

Общие права

Для начала выберем права для всего проекта и укажем их в settings.py:

```
REST_FRAMEWORK = {
    'DEFAULT_PERMISSION_CLASSES': [
        'rest_framework.permissions.DjangoModelPermissionsOrAnonReadOnly'
    ],
...
}
```

/library/settings.py

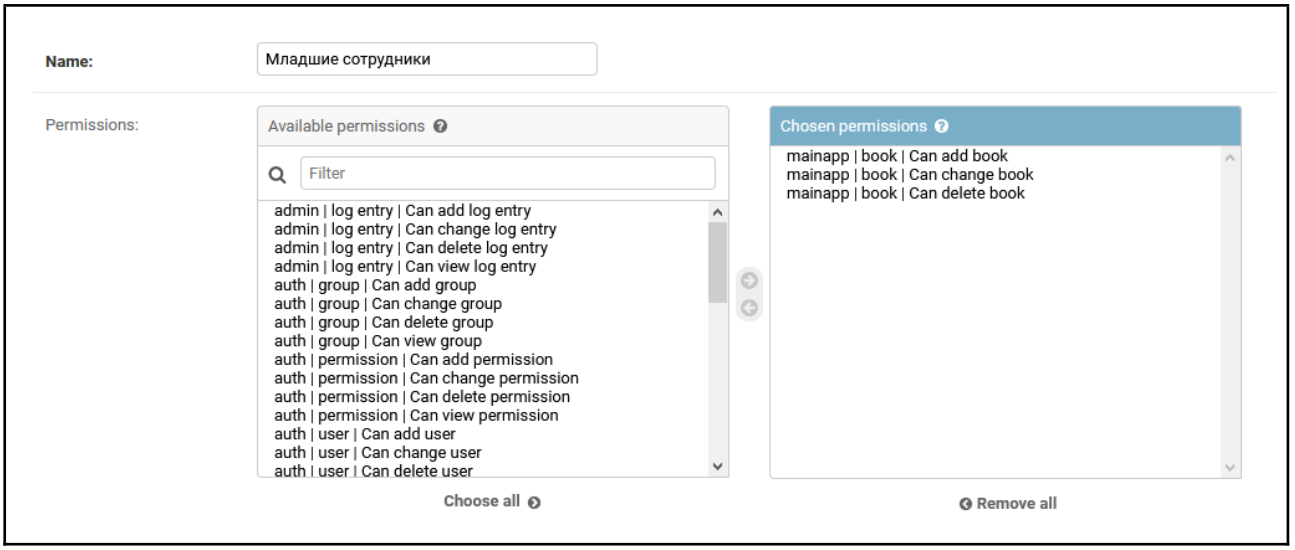
Мы выбрали DjangoModelPermissionsOrAnonReadOnly, так как у нас много видов пользователей. В такой ситуации удобно объединять их в группы и использовать права групп (DjangoModelPermissions позволяет это делать).

AnonReadOnly даёт доступ на просмотр для любых пользователей, что актуально для нашей задачи.

Создание групп и выдача прав

В стандартной админке Django можно создавать группы и выдавать им права. Для этого:

- нужно зайти в стандартную админку в раздел «Группы»;
- выбрать «Создать группу»;
- ввести название группы и выбрать права этой группы.

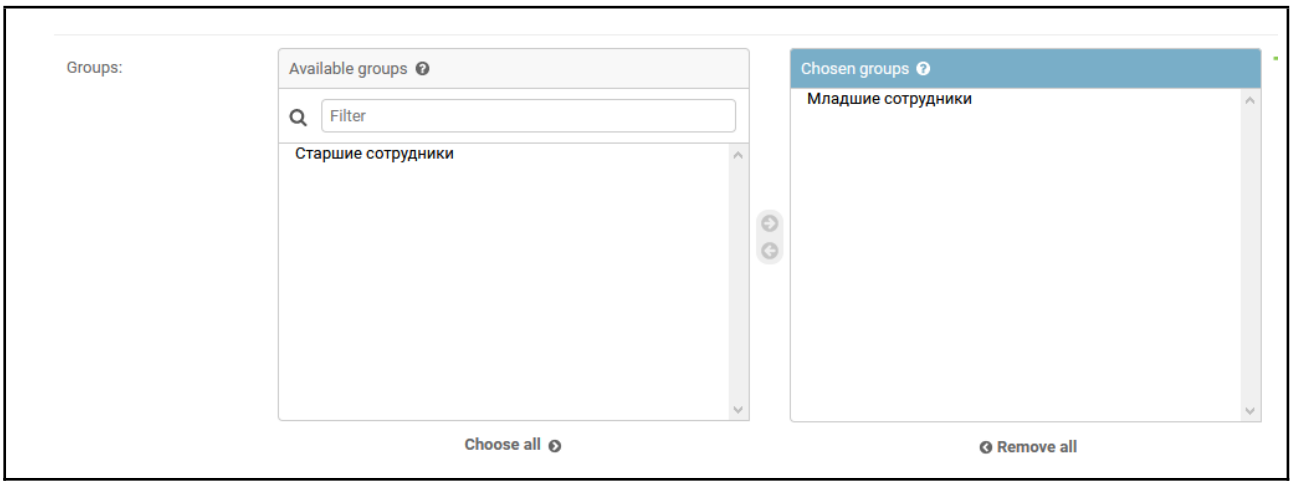


На скриншоте мы создали группу «Младшие сотрудники» и выдали им права на добавление (add), изменение (change), удаление (delete) модели Book.

По аналогии создадим группу «Старшие сотрудники» и выдадим такие же права на модель Book и Author.

Все пользователи, входящие в группу, будут наделены этими правами. Пользователь может включаться в несколько групп.

Для добавления пользователя в группу переходим в раздел пользователи (Users) в админке и выбираем нужные группы:



На скриншоте мы включили пользователя в группу младших сотрудников.

Проверка работы прав

После того как созданы пользователи и группы, проверим, что всё работает верно. Это можно сделать как в браузере, так и отправить запрос, например, с помощью библиотеки requests.

Пользователи, которые не входят ни в одну группу, и гости не смогут создавать книги (Book). В браузере форма для создания будет недоступна, а ответ на запрос вернёт 403 или 302 код.

Пользователи, входящие только в группу «Младшие сотрудники», смогут создавать книги (Book). Но они не создадут авторов (Author). Старшие сотрудники смогут создавать как книги (Book), так и авторов (Author).

Авторизация

Введение

После настройки прав нужно настроить авторизацию. Интерфейс, который нам предоставляет DRF и стандартная админка Django, подходит только для внутреннего использования. При работе стороннего пользователя на нашем сайте нужно его идентифицировать, понимать, кто он и какими правами обладает.

В Django пользователь хранился в сессии, но REST API не хранит состояния. Поэтому каждый запрос уникален и должен содержать данные о пользователе.

DRF предоставляет нам большие возможности для настройки авторизации на стороне сервера. Далее мы подробно рассмотрим основные из них.

Основные варианты авторизации

DRF предоставляет несколько вариантов авторизации, а также возможность использовать сторонние библиотеки и создавать свои. Рассмотрим основные варианты, которые удобно использовать для большинства проектов.

Базовая авторизация. BasicAuthentication

В заголовках запроса передаются логин и пароль пользователя. Этот вид авторизации простой и используется довольно часто. Его плюс — простота, так как у пользователя изначально есть логин и пароль, и ничего больше не требуется. Минус — безопасность, так как приходится в каждый запрос прикладывать данные пользователя, которые могут быть перехвачены и расшифрованы.

Авторизация по токену. TokenAuthentication

В заголовках запроса передаётся специальный токен авторизации. Это позволяет не передавать открыто логин и пароль. При перехвате или потере токена его можно пересоздать. Минус — опции по созданию и хранению токенов пользователей.

Авторизация в сессии. SessionAuthentication

Обычно дополняет какой-либо другой вид авторизации. Пользователь сохраняется в сессии и может запомниться на некоторое время. Такой вид авторизации не подходит для REST API, которое не хранит в себе состояния.

Настройка авторизации в проекте

Для настройки авторизации обычно достаточно:

- в настройках проекта указать нужные виды авторизации;
- для авторизации по токену создать таблицы для хранения токена и возможность его генерации.

В проект добавим все основные виды авторизации:

- базовую — для возможности зайти с логином и паролем;
- по токену — для работы на стороне клиента и для удобной и безопасной работы с API;
- в сессии — для работы внутренней части сайта, стандартной админки и DRF в браузере.

В settings.py укажем нужные классы авторизации:

```
REST_FRAMEWORK = {
    ...
    'DEFAULT_AUTHENTICATION_CLASSES': [
        'rest_framework.authentication.BasicAuthentication',
        'rest_framework.authentication.SessionAuthentication',
        'rest_framework.authentication.TokenAuthentication',
    ]
...
}
```

/library/settings.py

После этого Базовая авторизация и авторизация в сессии уже будут работать.

Для авторизации по токену сделаем ещё несколько действий.

В INSTALLED_APPS подключим приложение rest_framework.authtoken:

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

/library/settings.py

Затем сделаем миграции:

```
python manage.py migrate
```

/terminal

Теперь у нас будет модель для хранения токена и таблица с соответствием пользователя (User) и его токена (Token).

Далее в urls.py добавим адрес для получения токена пользователя:

```
...
from rest_framework.authtoken import views
...

urlpatterns = [
    ...
    path('api-token-auth/', views.obtain_auth_token)
]
```

```
/library/urls.py
```

При отправке post-запроса на этот адрес с указанием username и password вернётся токен авторизации. Его можно прикладывать ко всем последующим запросам.

Рассмотрим, как получить токен с помощью библиотеки requests:

```
import requests


response = requests.post('http://127.0.0.1:8000/api-token-auth/', data={'username': 'big', 'password': 'big123456'})

print(response.status_code) # {'token': '2efa08beed5727856319740df3747df4e0a3655e'}
print(response.json()) # {'token': '2efa08beed5727856319740df3747df4e0a3655e'}
```

Итоги

В этом занятии мы рассмотрели систему прав и авторизации, встроенную в DRF. Чтобы лучше понять, как она работает, на следующем занятии добавим авторизацию в клиентскую часть проекта.

Глоссарий

Авторизация (англ. authorization «разрешение; уполномочивание») — предоставление определённому лицу или группе лиц прав на выполнение конкретных действий; а также процесс проверки (подтверждения) данных прав при попытке выполнения этих действий. Часто можно услышать выражение, что какой-то человек «авторизован» для выполнения операции. Это значит, что он имеет на неё право.

Дополнительные материалы

1. Авторизация DRF.
2. Права DRF.
3. JWT-статья.

Используемые источники

1. Авторизация DRF.
2. Права DRF.

Практическое задание

Добавить права и авторизацию.

В этой самостоятельной работе тренируем умения

- использовать систему прав;
- использовать систему авторизации.

Зачем?

Чтобы использовать права и авторизацию в своих проектах.

Последовательность действий

1. В проекте реализовать систему прав.

   Есть 3 вида пользователей: администраторы, разработчики, владельцы проектов.

   - администраторы могут всё;
   - разработчики имеют все права на модель ToDo, могут просматривать модели Project и User;
   - владельцы проектов имеют права на просмотр модели User и все права на модель Project и ToDo.

2. Добавить в проект базовую авторизацию.

3. Добавить в проект авторизацию по токену.

4. (Задание со *) Добавить в проект авторизацию по JWT токену: ссылка.