DJANGO *for* APIs

Build web APIs with Python & Django

WILLIAM S. VINCENT

Introduction + các lệnh hay dùng

Tag youtube : django api , pipenv , startapp , createsuperuser , djangorestframework , Django restframework , django-allauth , django-cors-headers , virtualenv , API RESTful , Django REST Framework, crud Django , Django login logout , Django permission , django authentication, djangorestframework, token authentication, Django , install django with pip, virtualenv, web application development, pip, django, beginners tutorial, install python

cd ~/Desktop
mkdir blog
cd blog
pip install virtualenv # không dùng

pip install pipenv
pipenv install django==4.0
pipenv shell
pipenv run
pipenv lock

django-admin startproject blog_project .
python manage.py startapp blog
python manage.py createsuperuser
python manage.py migrate
python manage.py runserver

thư viện :
pipenv install django-crispy-forms==1.7.2
pip install crispy-bootstrap5
pipenv install djangorestframework
pipenv install django-cors-headers==3.4.0
pipenv install dj-rest-auth
pipenv install django-allauth
pipenv install pyyaml, uritemplate
pipenv install drf-yasg

Internet được cung cấp bởi các API RESTful. Đằng sau hậu trường ngay cả nhiệm vụ trực tuyến đơn giản nhất cũng liên quan đến nhiều máy tính tương tác với nhau .

Trong cuốn sách này, bạn sẽ học cách xây dựng nhiều API web RESTful với độ phức tạp ngày càng tăng từ đầu bằng cách sử dụng Django [1] và Django REST Framework[2], một trong những cách phổ biến và có thể tùy chỉnh nhất để xây dựng API web, được sử dụng bởi nhiều công ty công nghệ lớn nhất trên thế giới bao gồm Insta- gram, Mozilla, Pinterest và Bitbucket. Cách tiếp cận này cũng rất phù hợp với người mới bắt đầu vì phương pháp tiếp cận "bao gồm pin" của Django che giấu

phần lớn sự phức tạp và rủi ro bảo mật tiềm ẩn liên quan đến việc tạo bất kỳ trang web nào API.

Why this book

Cuốn sách này là hướng dẫn mà tôi ước tồn tại khi bắt đầu với Django REST Framework.

Chương 1 bắt đầu bằng phần giới thiệu ngắn gọn về các API web và giao thức HTTP. Trong **Chương 2**, chúng tôi xem xét sự khác biệt giữa Django truyền thống và Django REST Framework bằng cách xây dựng một trang web sách *Thư viện* và sau đó thêm một API vào đó. Sau đó, trong **Chương 3-4**, chúng ta xây dựng một API *Todo* và kết nối nó với front-end React. Quá trình tương tự có thể được sử dụng để kết nối bất kỳ front-end chuyên dụng nào (web, iOS, Android, máy tính để bàn hoặc khác) với back-end API web.

Trong **Chương 5-9**, chúng tôi xây dựng một API Blog sẵn sàng sản xuất bao gồm chức năng CRUD đầy đủ. Chúng tôi cũng đề cập đến các quyền chuyên sâu, xác thực người dùng, bộ xem, bộ định tuyến, tài liệu, v.v.

Chapter 1: Web APIs

Status Codes

Khi trình duyệt web của bạn đã thực hiện Yêu cầu HTTP trên URL, không có gì đảm bảo mọi thứ sẽ thực sự hoạt động! Do đó, có một danh sách khá dài các Mã trạng thái HTTP [19] có sẵn để đi kèm với mỗi phản hồi HTTP.

Bạn có thể biết *loại* mã trạng thái chung dựa trên hệ thống sau:

- 2xx Success - Hành động mà khách hàng yêu cầu đã được nhận , hiểu và chấp nhận
- 3xx Redirection - URL được yêu cầu đã di chuyển
- 4xx Client Error - đã xảy ra lỗi, thường là yêu cầu URL không hợp lệ của máy khách
- 5xx Server Error - the server failed to resolve a request

Không cần phải ghi nhớ tất cả các mã trạng thái có sẵn. Với thực hành, bạn sẽ trở nên quen thuộc với những cái phổ biến nhất như 200 (OK), 201 (Created), 301 (Moved Permanently), 404 (Not

Điều quan trọng cần nhớ là, nói chung, chỉ có bốn kết quả tiềm năng cho bất kỳ yêu cầu HTTP nhất định nào: nó hoạt động (2xx), nó được chuyển hướng bằng cách nào đó (3xx), máy khách mắc lỗi (4xx) hoặc máy chủ đã mắc lỗi (5xx)).

Chapter 2: Library Website and API

Luồng công việc :

```
# pip => tạo config => startapp => sửa config_setting => viết model => sửa book/admin => sửa book.views => sửa coonfig/urls => thêm book/url => thêm template
# rest : pip => sửa config/setting => thêm start app=> sửa config/setting => sửa lại config/url => thêm api/url và sửa => sửa api/view => thêm và sửa api/serializers => test curl/8000
```

Traditional Django

pip install pipenv

Command Line

```
$ cd ~/Desktop
```

Command Line

```
$ mkdir library && cd library
$ pipenv install django==4.0
$ pipenv shell
(library) $
```

Pipenv tạo một Pipfile và một Pipfile.lock trong thư mục hiện tại của chúng tôi. (thư viện) trong ngoặc đơn trước dòng lệnh cho thấy môi trường ảo của chúng ta đang hoạt động.

Command Line

```
(library) $ django-admin startproject config .
```

Command Line

```
(library) $ tree
.
├── Pipfile
├── Pipfile.lock
├── config
│   ├── __init__.py
```

```
│   ├── asgi.py
│   ├── settings.py
│   ├── urls.py
│   └── wsgi.py
└── manage.py

1 directory, 8 files
```

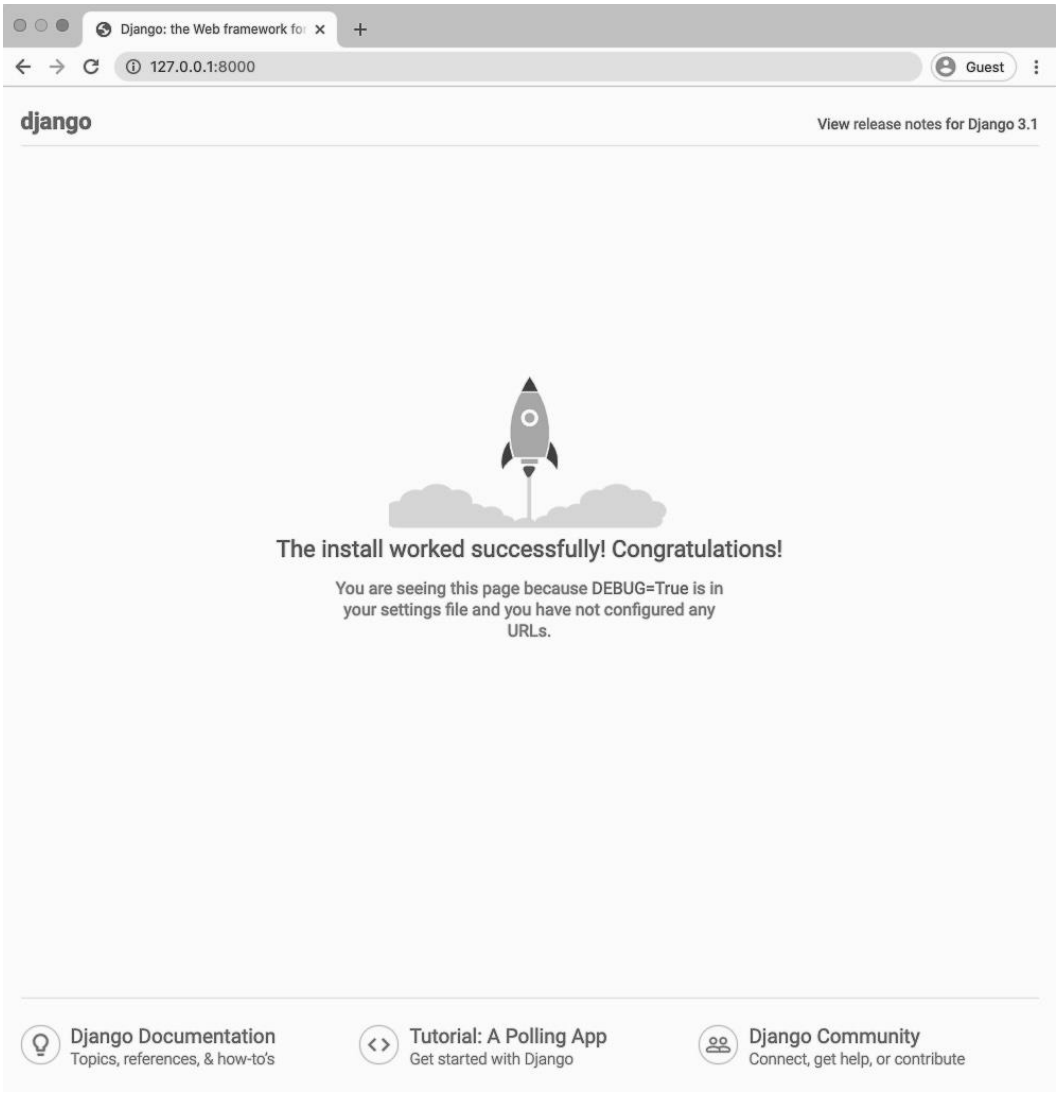
- `__init__.py` là một cách Python để coi một thư mục như một gói; nó trống rỗng
- `asgi.py` là viết tắt của Giao *diện cổng máy chủ không đồng bộ* và là một tùy chọn mới trong Django 3.0+
- `settings.py` chứa tất cả các cấu hình cho dự án của chúng tôi
- `urls.py` kiểm soát các tuyến URL cấp cao nhất
- `wsgi.py` là viết tắt của *Web Server Gateway Interface* và giúp Django phục vụ các trang web cuối cùng
- `manage.py` thực thi các lệnh Django khác nhau như chạy máy chủ web cục bộ hoặc tạo một ứng dụng mới.

Command Line

```
(library) $ python manage.py migrate
(library) $ python manage.py runserver
```

Open a web browser to http://127.0.0.1:8000/[25] to confirm our project is successfully installed.

[25] http://127.0.0.1:8000/



Django welcome page

First app

Command Line

```
(library) $ python manage.py startapp books
```

Command Line

```
(library) $ tree
.
├── Pipfile
├── Pipfile.lock
├── books
│   ├── __init__.py
│   ├── admin.py
│   ├── apps.py
│   ├── migrations
│   │   └── __init__.py
│   ├── models.py
│   ├── tests.py
│   └── views.py
├── config
│   ├── __init__.py
│   ├── asgi.py
│   ├── settings.py
│   ├── urls.py
│   └── wsgi.py
├── db.sqlite3
└── manage.py
```

Each app has a `__init__.py` file identifying it as a Python package. There are 6 new files created:

- `admin.py` là một tệp cấu hình cho ứng dụng Django Admin tích hợp
- `apps.py` là một tệp cấu hình cho chính ứng dụng
- the `migrations/` thư mục lưu trữ các tệp di chuyển để thay đổi cơ sở dữ liệu
- `models.py`là nơi chúng tôi xác định các mô hình cơ sở dữ liệu của mình
- `tests.py` dành cho các thử nghiệm dành riêng cho ứng dụng của chúng tôi

  `views.py` là nơi chúng ta xử lý logic request/response cho ứng dụng web của mình
-

Code

```
# config/settings.py
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',

    # Local
    'books', # new
]
```

Models

In your text editor, open up the file `books/models.py` and update it as follows:

Code

```
# books/models.py
from django.db import models

class Book(models.Model):
    title = models.CharField(max_length=250)
    subtitle = models.CharField(max_length=250)
    author = models.CharField(max_length=100)
    isbn = models.CharField(max_length=13)

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

[26] https://www.isbn-international.org/content/what-isbn

Command Line

```
(library) $ python manage.py makemigrations books
```

Admin

Command Line

```
(library) $ python manage.py createsuperuser
```

Code

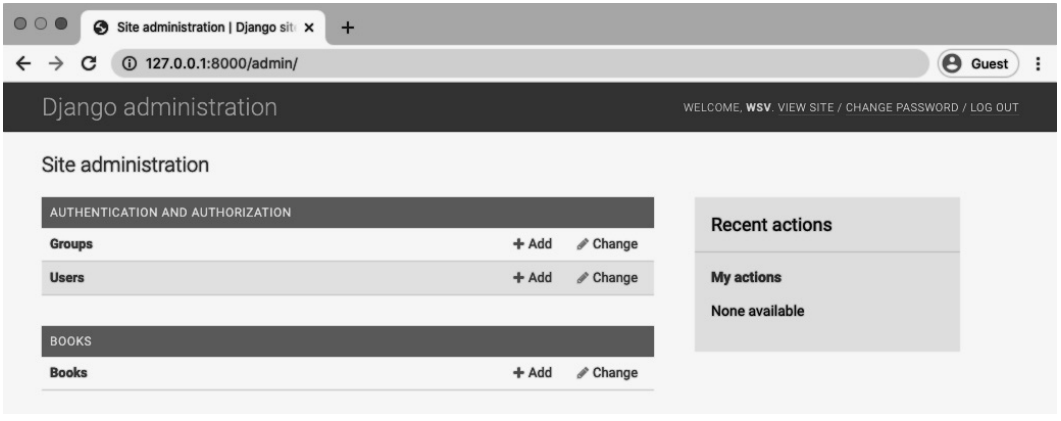
```
# books/admin.py
from django.contrib import admin
from .models import Book

admin.site.register(Book)
```

That's all we need! Start up the local server again.

Command Line

```
(library) $ python manage.py runserver
```

Navigate to `http://127.0.0.1:8000/admin` and log in. You will be redirected to the admin homepage.



Views

Code

```
# books/views.py
from django.views.generic import ListView
from .models import Book


class BookListView(ListView):
    model = Book
    template_name = 'book_list.html'
```

URLs

Code

```
# config/urls.py
from django.contrib import admin
from django.urls import path, include # new

urlpatterns = [
    path('admin/', admin.site.urls),
    path('', include('books.urls')), # new
]
```

Command Line

```
(library) $ touch books/urls.py
```

Code

```
# books/urls.py
from django.urls import path
from .views import BookListView

urlpatterns = [
    path('', BookListView.as_view(), name='home'),
]
```

Command Line

```
(library) $ mkdir books/templates
(library) $ mkdir books/templates/books
(library) $ touch books/templates/books/book_list.html
```

Now update the template file.

HTML

```
<!-- books/templates/books/book_list.html -->
```

Webpage

Command Line

```
(library) $ python manage.py runserver
```

Navigate to the homepage which is at `http://127.0.0.1:8000/`.

Book web page

Django REST Framework

Django REST Framework được thêm vào giống như bất kỳ ứng dụng bên thứ ba nào khác. Đảm bảo thoát khỏi máy chủ cục bộ Control + c nếu nó vẫn đang chạy. Sau đó, trên dòng lệnh gõ bên dưới.

Command Line

```
(library) $ pipenv install djangorestframework~=3.11.0
```

Code

```
# config/settings.py
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
```

```
    # 3rd party
    'rest_framework', # new

    # Local
    'books',
]
```

Cuối cùng, API của chúng tôi sẽ hiển thị một điểm cuối duy nhất liệt kê tất cả sách trong JSON. Vì vậy, chúng ta sẽ cần một tuyến URL mới, một chế độ xem mới và một tệp serializer mới (sẽ sớm có thêm về điều này).

Command Line

```
(library) $ python manage.py startapp api
```

Then add it to `INSTALLED_APPS`.

Code

```
# config/settings.py
INSTALLED_APPS = [
    # Local
    'books.apps.BooksConfig',
    'api.apps.ApiConfig', # new

    # 3rd party
    'rest_framework',

    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
]
```

URLs

Code

```
# config/urls.py
from django.contrib import admin
from django.urls import path, include

urlpatterns = [
    path('admin/', admin.site.urls),
    path('api/', include('api.urls')), # new
    path('', include('books.urls')),
]
```

Command Line

```
(library) $ touch api/urls.py
```

And update it as follows:

Code

```
# api/urls.py
from django.urls import path
from .views import BookAPIView

urlpatterns = [
    path('', BookAPIView.as_view()),
]
```

Views

Tiếp theo là tệp views.py của chúng ta dựa trên các chế độ xem lớp chung được tích hợp sẵn của Django REST Framework. Những điều này cố tình bắt chước các quan điểm dựa trên lớp chung của Django truyền thống ở định dạng, nhưng chúng **không** giống nhau.

Để tránh nhầm lẫn, một số nhà phát triển sẽ gọi tệp chế độ xem API apiviews.py hoặc api.py. Cá nhân tôi, khi làm việc trong một ứng dụng api chuyên dụng, tôi không thấy khó hiểu khi chỉ gọi tệp chế độ xem Django REST Framework views.py nhưng ý kiến khác nhau về điểm này.

Within our `views.py` file, update it to look like the following:

Code

```
# api/views.py
from rest_framework import generics
from books.models import Book
from .serializers import BookSerializer


class BookAPIView(generics.ListAPIView):
    queryset = Book.objects.all()
    serializer_class = BookSerializer
```

On the top lines we import Django REST Framework's `generics` class of views, the models from our `books` app, and `serializers` from our `api` app (we will make the serializers next).

Sau đó, chúng tôi tạo một `BookAPIView` that uses `ListAPIView` để tạo điểm cuối chỉ đọc cho tất cả các phiên bản sách. Có rất nhiều quan điểm chung có sẵn và chúng ta sẽ khám phá thêm về chúng trong các chương sau.

Hai bước duy nhất cần thiết trong chế độ xem của chúng tôi là chỉ định `queryset` đó là tất cả các cuốn sách có sẵn, và sau đó là`serializer_class` cái sẽ là`BookSerializer`.

Serializers

Trình tuần tự hóa chuyển dữ liệu sang định dạng dễ sử dụng qua internet, thường là JSON và được hiển thị tại điểm cuối API. Chúng tôi cũng sẽ đề cập sâu hơn đến các bộ nối tiếp hóa và JSON trong các chương sau. Hiện tại, tôi muốn chứng minh việc tạo serializer với Django REST Framework để chuyển đổi các mô hình Django sang JSON dễ dàng như thế nào.

Make a `serializers.py` file within our `api` app.

Command Line

```
(library) $ touch api/serializers.py
```

Code

```
# api/serializers.py
from rest_framework import serializers
from books.models import Book


class BookSerializer(serializers.ModelSerializer):
    class Meta:
        model = Book
        fields = ('title', 'subtitle', 'author', 'isbn')
```

Ở dòng trên cùng, chúng tôi nhập lớp bộ nối tiếp hóa của Django REST Framework và mô hình Sách từ ứng dụng sách của chúng tôi. Chúng tôi mở rộng ModelSerializer của Django REST Framework thành một BookSerializer lớp chỉ định mô hình cơ sở dữ liệu của chúng tôi Sách và các trường cơ sở dữ liệu mà chúng tôi muốn hiển thị: tiêu đề, phụ đề, tác giả và isbn.

cURL

Chúng tôi muốn xem điểm cuối API của chúng tôi trông như thế nào. Chúng tôi biết nó sẽ trả về JSON tại URL http://127.0.0.1:8000/api/ [33]. Hãy đảm bảo rằng máy chủ Django cục bộ của chúng tôi đang chạy:

Command Line

```
(library) $ python manage.py runserver
```

Chúng ta có thể sử dụng chương trình cURL[34] phổ biến để thực thi các yêu cầu HTTP thông qua dòng lệnh. Tất cả những gì chúng ta cần cho một yêu cầu GET cơ bản, nó để chỉ định curl và URL mà chúng ta muốn gọi.

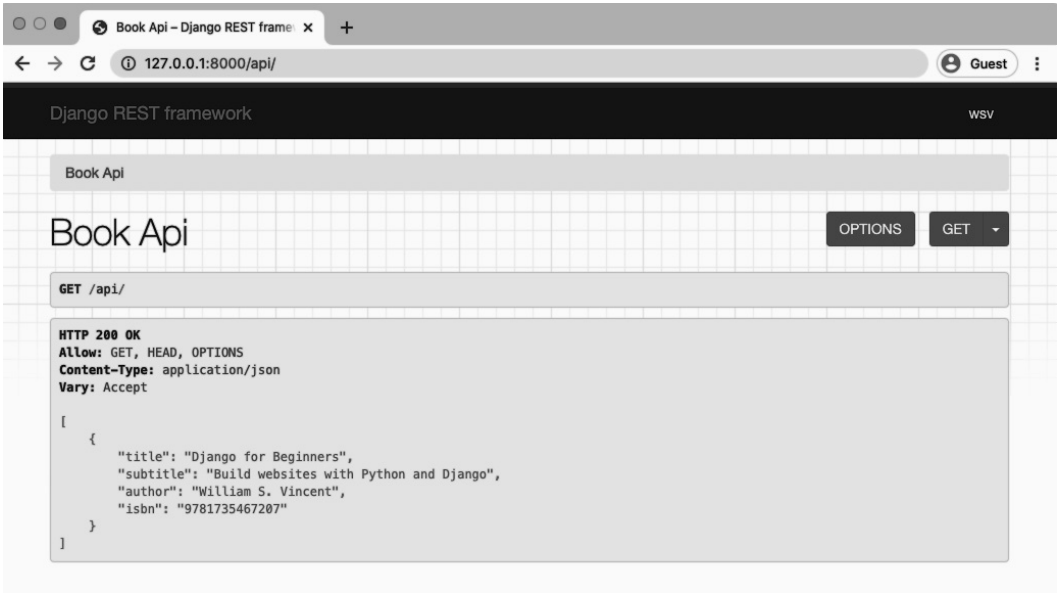
Command Line

```
$ curl http://127.0.0.1:8000/api/
[
   {
      "title":"Django for Beginners",
      "subtitle":"Build websites with Python and Django",
      "author":"William S. Vincent",
      "isbn":"9781735467207"
```

```
    }
]
```

Browsable API

With the local server still running in the first command line console, navigate to our API endpoint in the web browser at `http://127.0.0.1:8000/api/`.



Book API

This is what the raw JSON from our API endpoint looks like. I think we can agree the Django REST Framework version is more appealing.

Chapter 3: Todo API

```
# Luồng công việc : initial setup => startproject và startapp = > edit
todos/models =>  migrate => edit todos/admin => runserver =>  pip
djangorestframework => edit config/setting => edit config url => config
todos/url => edit todo/serializer => edit todos/views => runserver =>
config/setting
```

Trong suốt hai chương tiếp theo, chúng ta sẽ xây dựng back-end *Todo* API và sau đó kết nối nó với front-end React. Chúng tôi đã tạo API đầu tiên của mình và xem xét cách HTTP và REST hoạt động trong bản tóm tắt nhưng vẫn có khả năng bạn chưa "khá" xem tất cả kết hợp với nhau như thế nào. Đến cuối hai chương này, bạn sẽ làm được.

Diagram

```
todo
|   ├──frontend
|       ├──React...
|   ├──backend
|       ├──Django...
```

Chương này tập trung vào back-end và Chương 4 ở front-end.

Initial Set Up

Open a new command line console and enter the following commands:

Command Line

```
$ cd ~/Desktop
$ cd code
$ mkdir todo && cd todo
```

Trong thư mục todo này sẽ là các thư mục backend và frontend của chúng ta. Hãy tạo backend thư mục, cài đặt Django và kích hoạt một môi trường ảo mới.

Command Line

```
$ mkdir backend && cd backend
$ pipenv install django~=3.1.0
$ pipenv shell
```

Command Line

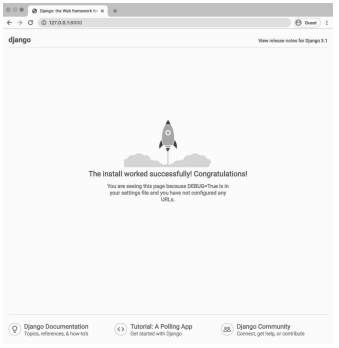
```
(backend) $ django-admin startproject config .
(backend) $ python manage.py startapp todos
(backend) $ python manage.py migrate
```

Code

```
# config/settings.py
INSTALLED_APPS = [
    'django.contrib.admin',
```

```
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',

    # Local
    'todos', # new
]
```

If you run `python manage.py runserver` on the command line now and navigate in your web browser to `http://127.0.0.1:8000/` you can see our project is successfully installed



Models

Code

```
# todos/models.py
from django.db import models


class Todo(models.Model):
    title = models.CharField(max_length=200)
    body = models.TextField()

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

Command Line

```
(backend) $ python manage.py makemigrations todos
Migrations for 'todos':
  todos/migrations/0001_initial.py
```

```
    - Create model Todo
(backend) $ python manage.py migrate
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, sessions, todos
Running migrations:
  Applying todos.0001_initial... OK
```

```
  python manage.py makemigrations
```

Code

```
# todos/admin.py
from django.contrib import admin
from .models import Todo

admin.site.register(Todo)
```

Command Line

```
(backend) $ python manage.py createsuperuser
```

Command Line

```
(backend) $ python manage.py runserver
```

Django REST Framework

Command Line

```
(backend) $ pipenv install djangorestframework~=3.11.0
```

Code

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',

    # 3rd party
    'rest_framework', # new

    # Local
    'todos',
]
```

```
# new
REST_FRAMEWORK = {
    'DEFAULT_PERMISSION_CLASSES': [
        'rest_framework.permissions.AllowAny',
    ]
}
```

Django REST Framework có một danh sách dài các cài đặt mặc định được đặt ngầm. Bạn có thể xem danh sách đầy đủ tại đây[36]. AllowAny là một trong số đó có nghĩa là khi chúng ta đặt nó một cách rõ ràng, như chúng ta đã làm ở trên, hiệu ứng hoàn toàn giống như khi chúng ta không có cấu hình DEFAULT_PERMISSION_CLASSES cài.

Thay vào đó, chúng tôi sẽ cập nhật ba tệp Django REST Framework cụ thể để chuyển đổi mô hình cơ sở dữ liệu của chúng tôi thành API web: urls.py, views.py và serializers.py.

URLs

Code

```
# config/urls.py
from django.contrib import admin
from django.urls import include, path # new


urlpatterns = [
    path('admin/', admin.site.urls),
    path('api/', include('todos.urls')), # new
]
```

Command Line

```
(backend) $ touch todos/urls.py
```

Code

```
# todos/urls.py
from django.urls import path
from .views import ListTodo, DetailTodo

urlpatterns = [
    path('<int:pk>/', DetailTodo.as_view()),
    path('', ListTodo.as_view()),
```

```
]
```

Lưu ý rằng chúng tôi đang tham chiếu đến hai chế độ xem — ListTodo và DetailTodo — mà chúng tôi vẫn chưa tạo. Nhưng việc định tuyến hiện đã hoàn tất. Sẽ có một danh sách tất cả các việc cần làm ở chuỗi trống '', nói cách khác là tại api/. Và mỗi việc cần làm riêng lẻ sẽ có sẵn tại khóa chính của nó, đây là một giá trị mà Django tự động đặt trong mọi bảng cơ sở dữ liệu. Mục đầu tiên là 1, mục thứ hai là 2, v.v. Do đó, todo đầu tiên của chúng ta cuối cùng sẽ được đặt tại api/1/ .

Serializers

Hãy xem xét vị trí của chúng tôi cho đến nay. Chúng tôi bắt đầu với một dự án và ứng dụng Django truyền thống, nơi chúng tôi tạo mô hình cơ sở dữ liệu và thêm dữ liệu. Sau đó, chúng tôi đã cài đặt Django REST Framework và định cấu hình URL của chúng tôi. Bây giờ chúng ta cần chuyển đổi dữ liệu của mình, từ các mô hình, thành JSON sẽ được xuất ra tại các URL. Do đó, chúng ta cần một bộ nối tiếp hóa.

Django REST Framework đi kèm với lớp serializers tích hợp sẵn mạnh mẽ mà chúng ta có thể nhanh chóng mở rộng với một lượng mã nhỏ. Đó là những gì chúng tôi sẽ làm ở đây.

Command Line

```
(backend) $ touch todos/serializers.py
```

Code

```
# todos/serializers.py
from rest_framework import serializers
from .models import Todo


class TodoSerializer(serializers.ModelSerializer):
    class Meta:
        model = Todo
        fields = ('id', 'title', 'body',)
```

Ở trên cùng, chúng tôi đã nhập các bộ nối tiếp hóa từ Django REST Framework cũng như tệp models.py của chúng tôi . Tiếp theo chúng ta tạo một class TodoSerializer. Định dạng ở đây **rất** giống với cách chúng ta tạo các lớp hoặc biểu mẫu mô hình trong chính Django. Chúng tôi đang chỉ định mô hình nào sẽ sử dụng và các trường cụ thể trên đó mà chúng tôi muốn hiển thị. Hãy nhớ rằng id được tạo tự động bởi Django nên chúng ta không phải định nghĩa nó trong mô hình Todo của mình nhưng chúng ta sẽ sử dụng nó trong chế độ xem chi tiết của mình.

Và đó là nó. Django REST Framework giờ đây sẽ chuyển đổi dữ liệu của chúng ta một cách kỳ diệu thành JSON hiển thị các trường cho id, title và body từ mô hình Todo của chúng ta.

Điều cuối cùng chúng ta cần làm là cấu hình tệp views.py của chúng ta.

Views

Code

```
# todos/views.py
from rest_framework import generics
from .models import Todo
from .serializers import TodoSerializer


class ListTodo(generics.ListAPIView):
    queryset = Todo.objects.all()
    serializer_class = TodoSerializer


class DetailTodo(generics.RetrieveAPIView):
    queryset = Todo.objects.all()
    serializer_class = TodoSerializer
```

Ở trên cùng, chúng tôi nhập các chế độ xem chung của Django REST Framework và cả models.py của chúng tôi và `serializers.py` files.

Hãy nhớ lại từ tệp todos / urls.py của chúng tôi rằng chúng tôi có hai tuyến đường và do đó có hai chế độ xem riêng biệt. Chúng ta sẽ sử dụng ListAPIView để hiển thị tất cả các todos và RetrieveAPIView để hiển thị một phiên bản model duy nhất.

Những độc giả sắc sảo sẽ nhận thấy rằng có một chút dư thừa trong mã ở đây. Về cơ bản, chúng tôi lặp lại `queryset` and `serializer_class` đối với mỗi chế độ xem, mặc dù chế độ xem chung được mở rộng là khác nhau. Sau này trong cuốn sách, chúng ta sẽ tìm hiểu về các bộ chế độ xem và bộ định tuyến giải quyết vấn đề này và cho phép chúng ta tạo cùng một chế độ xem API và URL với ít mã hơn nhiều.

Consuming the API

Theo truyền thống, việc sử dụng một API là một thách thức. Đơn giản là không có hình ảnh trực quan tốt cho tất cả thông tin có trong nội dung và tiêu đề của một yêu cầu hoặc phản hồi HTTP nhất định.

Thay vào đó, hầu hết các nhà phát triển đã sử dụng ứng dụng khách HTTP dòng lệnh như

cURL[39], mà chúng ta đã thấy trong chương trước hoặc HTTPie[40].

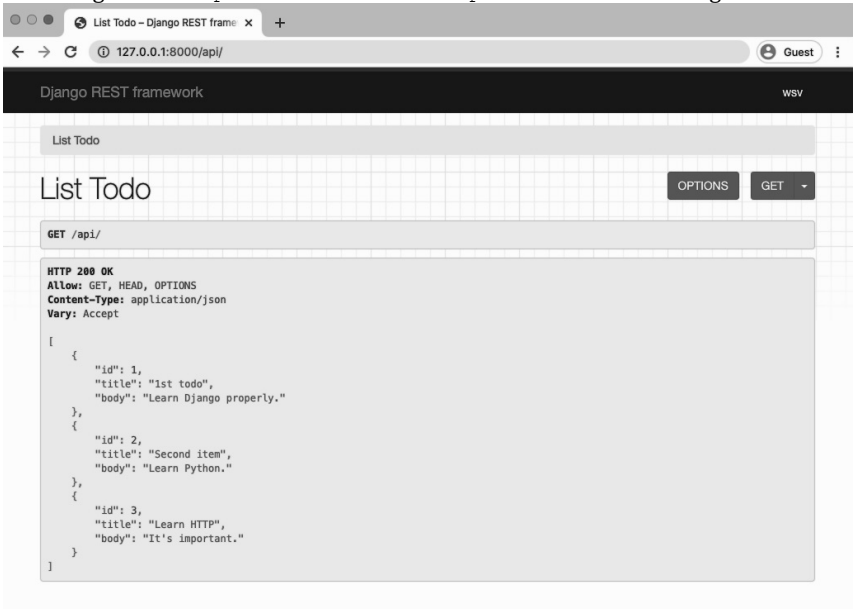
Vào năm 2012, sản phẩm phần mềm của bên thứ ba Postman đã được ra mắt và nó hiện được sử dụng bởi hàng triệu nhà phát triển trên toàn thế giới, những người muốn có một cách trực quan, giàu tính năng để tương tác với API.

Browsable API

Command Line

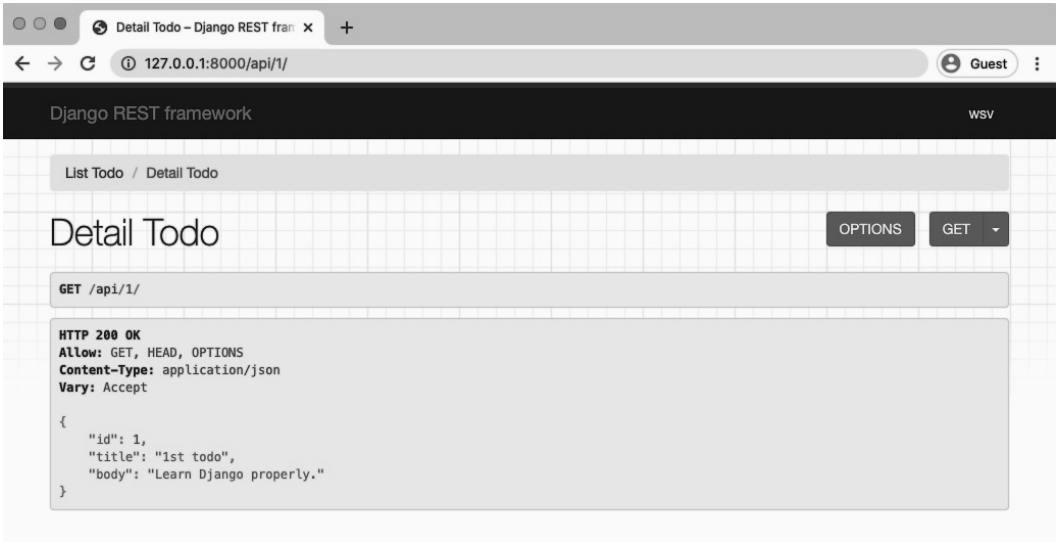
```
(backend) $ python manage.py runserver
```

Then navigate to `http://127.0.0.1:8000/api/` to see our working API list views endpoint.



API List

We also made a `DetailTodo` view for each individual model. This is known as an **instance** and is visible at `http://127.0.0.1:8000/api/1/`.



API Detail

CORS

Có một bước cuối cùng chúng ta cần làm và đó là giải quyết Chia sẻ tài nguyên nguồn gốc chéo (CORS) [42]. Bất cứ khi nào máy khách tương tác với API được lưu trữ trên một miền khác (mysite.com so với yoursite.com) hoặc cổng (localhost: 3000 vs localhost: 8000) có các vấn đề bảo mật tiềm ẩn.

Cụ thể, CORS yêu cầu máy chủ bao gồm các tiêu đề HTTP cụ thể cho phép máy khách xác định xem có nên cho phép các yêu cầu miền chéo hay không và khi nào.

Gói chúng tôi sẽ sử dụng là django-cors-headers[44], có thể dễ dàng thêm vào dự án hiện có của chúng tôi .

Đầu tiên thoát khỏi máy chủ của chúng tôi với Control + c và sau đó cài đặt tiêu đề django-cors với Pipenv.

Command Line

```
(backend) $ pipenv install django-cors-headers==3.4.0
```

Code

```
# config/settings.py
INSTALLED_APPS = [
    'django.contrib.admin',
```

```
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',

    # 3rd party
    'rest_framework',
    'corsheaders', # new

    # Local
    'todos',
]

MIDDLEWARE = [
    'django.middleware.security.SecurityMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
    'corsheaders.middleware.CorsMiddleware', # new
    'django.middleware.common.CommonMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
    'django.middleware.clickjacking.XFrameOptionsMiddleware',
]

# new
CORS_ORIGIN_WHITELIST = (
    'http://localhost:3000',
    'http://localhost:8000',
)
```

Điều rất quan trọng là corsheaders.middleware.CorsMiddleware xuất hiện trong lo- cation thích hợp. Đó là trên django.middleware.common.CommonMiddleware trong cài đặt MIDDLEWARE vì middleware được tải từ trên xuống dưới. Cũng lưu ý rằng chúng tôi đã đưa vào danh sách trắng hai tên miền: localhost: 3000 và localhost: 8000. Cái trước là cổng mặc định cho React, mà chúng ta sẽ sử dụng cho front-end của chúng ta trong chương tiếp theo. Cái sau là cổng Django mặc định.

Tests

Code

```
# todos/tests.py
from django.test import TestCase
from .models import Todo


class TodoModelTest(TestCase):
```

```
    @classmethod
    def setUpTestData(cls):
        Todo.objects.create(title='first todo', body='a body here')

    def test_title_content(self):
        todo = Todo.objects.get(id=1)
        expected_object_name = f'{todo.title}'
        self.assertEqual(expected_object_name, 'first todo')

    def  test_body_content(self):
        todo = Todo.objects.get(id=1)
        expected_object_name = f'{todo.body}'
        self.assertEqual(expected_object_name, 'a body here')
```

Command Line

```
(backend) $ python manage.py test
Creating test database for alias 'default'...
System check identified no issues (0 silenced).
..
----------------------------------------------------------------------
Ran 2 tests in 0.002s

OK
Destroying test database for alias 'default'...
```

Command Line

```
(backend) $ python manage.py runserver
```

Conclusion

Một điểm quan trọng trong ví dụ này là chúng tôi đã thêm các tiêu đề CORS và chỉ đặt rõ ràng các miền localhost: 3000 và localhost: 8000 để có quyền truy cập vào API của chúng tôi. Đặt tiêu đề CORS một cách chính xác là một điều dễ bị nhầm lẫn khi bạn lần đầu tiên bắt đầu xây dựng API.

Chapter 4: Todo React Front-end

Một API tồn tại để giao tiếp với một chương trình khác. Trong chương này, chúng ta sẽ sử dụng

API *Todo* của mình từ chương cuối cùng thông qua giao diện người dùng React[46] để bạn có thể thấy mọi thứ thực sự hoạt động cùng nhau như thế nào trong thực tế.

Vue[47], Angular[48], or Ember[49].

Install Node

Command Line

```
$ brew install node
```

Trên máy tính Windows có nhiều cách tiếp cận nhưng một cách phổ biến là sử dụng nvm-windows[54]. Kho lưu trữ của nó chứa các hướng dẫn cài đặt chi tiết, cập nhật.

Command Line

```
$ curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.35.3/\
install.sh | bash
```

or using Wget

Command Line

```
$ wget -qO- https://raw.githubusercontent.com/nvm-sh/nvm/v0.35.3/\
install.sh | bash
```

Then run:

Command Line

```
$ command -v nvm
```

Close your current command line console and open it again to complete installation.

Install React

cài đặt React chúng ta sẽ dựa vào npm[57] là một trình quản lý gói JavaScript. Giống như pipenv cho Python, npm làm cho việc quản lý và cài đặt nhiều gói phần mềm trở nên đơn giản hơn nhiều. Các phiên bản gần đây của npm cũng bao gồm npx[58], đây là một cách cải tiến để cài đặt các gói cục bộ mà không gây ô nhiễm không gian tên toàn cầu. Đó là cách được khuyến nghị để cài đặt React và những gì chúng ta sẽ sử dụng ở đây!

Đảm bảo rằng bạn đang ở đúng thư mục bằng cách điều hướng đến Màn hình nền (nếu trên máy

Mac) và sau đó là thư mục việc cần làm.

Command Line

```
$ cd ~/Desktop
$ cd todo
```

Create a new React app called `frontend`.

Command Line

```
$ npx create-react-app frontend
```

Your directory structure should now look like the following:

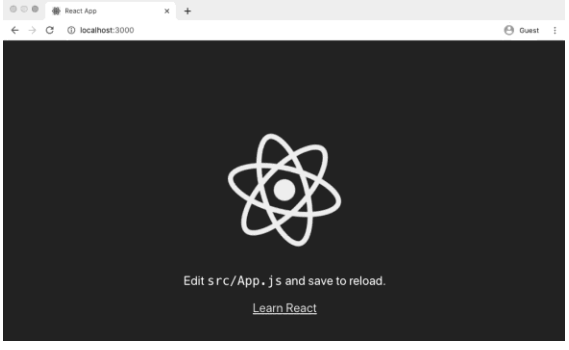
Diagram

```
todo
|   ├──frontend
|       ├──React...
|   ├──backend
|       ├──Django...
```

Command Line

```
$ cd frontend
$ npm start
```

If you navigate to `http://localhost:3000/` you will see the create-react-app default homepage.



React welcome page

Mock data

Nếu bạn quay lại điểm cuối API của chúng tôi, bạn có thể thấy JSON thô trong trình duyệt tại:

`http://127.0.0.1:8000/api/?format=json`

Code

```
[
  {
    "id":1,
    "title":"1st todo",
    "body":"Learn Django properly."
  },
  {
    "id":2,
    "title":"Second item",
    "body":"Learn Python."
  },
  {
    "id":3,
    "title":"Learn HTTP",
    "body":"It's important."
  }
]
```

Điều này được trả về bất cứ khi nào một yêu cầu GET được cấp cho điểm cuối API. Cuối cùng, chúng ta sẽ sử dụng API trực tiếp nhưng bước đầu tiên tốt là mock dữ liệu trước, sau đó cấu hình lệnh gọi API của chúng ta.

Tập tin duy nhất chúng ta cần cập nhật trong ứng dụng React là src/App.js. Hãy bắt đầu bằng cách mô phỏng dữ liệu API trong một danh sách có tên biến, đây thực sự là một mảng có ba giá trị.

Code

```
// src/App.js
import React, { Component } from 'react';

const list = [
  {
    "id":1,
    "title":"1st todo",
    "body":"Learn Django properly."
  },
  {
    "id":2,
    "title":"Second item",
    "body":"Learn Python."
  },
  {
    "id":3,
```

```
    "title":"Learn HTTP",
    "body":"It's important."
  }
]
```

Tiếp theo , chúng ta load list vào trạng thái của component rồi sử dụng JavaScript array method map()[59] để hiển thị tất cả các item.

Tôi đang cố tình di chuyển nhanh ở đây nên nếu bạn chưa bao giờ sử dụng React trước đây, chỉ cần sao chép code để bạn có thể thấy nó "sẽ" hoạt động như thế nào để kết nối front-end của React với back-end Django của chúng tôi.

Here's the complete code to include in the `src/App.js` file now.

Code

```
// src/App.js
import React, { Component } from 'react';

const list = [
  {
    "id":1,
    "title":"1st todo",
    "body":"Learn Django properly."
  },
  {
    "id":2,
    "title":"Second item",
    "body":"Learn Python."
  },
  {
    "id":3,
    "title":"Learn HTTP",
    "body":"It's important."
  }
]

class App extends Component {
  constructor(props) {
    super(props);
    this.state = { list };
  }
  render() {
    return (
      <div>
        {this.state.list.map(item => (
          <div key={item.id}>
            <h1>{item.title}</h1>
            <p>{item.body}</p>
```

```
          </div>
        ))}
      </div>
    );
  }
}

export default App;
```

Chúng tôi đã tải danh sách vào trạng thái của thành phần Ứng dụng, sau đó chúng tôi đang sử dụng map để lặp lại từng mục trong danh sách hiển thị tiêu đề và nội dung của từng mục. Chúng ta cũng đã thêm id làm key là một yêu cầu dành riêng cho React; id được Django tự động thêm vào mọi trường database cho chúng ta.

Bây giờ bạn sẽ thấy những việc cần làm của chúng tôi được liệt kê trên trang chủ ở http://localhost:3000/ mà không cần làm mới trang.

Dummy data

Lưu ý: Nếu bạn dành bất kỳ thời gian nào để làm việc với React, rất có thể đến một lúc nào đó bạn sẽ thấy thông báo lỗi sh: react-scripts: lệnh không tìm thấy trong khi chạy npm start. Đừng lo lắng.

Đây là một vấn đề rất, *rất phổ biến* trong phát triển JavaScript. Khắc phục sự cố thường là chạy cài đặt npm và sau đó thử npm bắt đầu lại. Nếu điều đó không hoạt động, sau đó xóa thư mục node_modules của bạn và chạy npm install. Điều đó giải quyết vấn đề 99% thời gian. Chào mừng bạn đến với :) phát triển JavaScript hiện đại.

Django REST Framework + React

Now let's hook into our *Todo* API for real instead of using the mock data in the `list` variable. In the other command line console our Django server is running and we know the API endpoint listing all todos is at `http://127.0.0.1:8000/api/`. So we need to issue a `GET` request to it.

There are two popular ways to make HTTP requests: with the built-in Fetch API[60] or with axios[61], which comes with several additional features. We will use `axios` in this example. Stop the React app currently running on the command line with `Control+c`. Then install `axios`.

Command Line

```
$ npm install axios
```

Start up the React app again using `npm start`.

Command Line

```
$ npm start
```

Then in your text editor at the top of the `App.js` file import Axios.

[60] https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API
[61] https://github.com/axios/axios

Code

```
// src/App.js
import React, { Component } from 'react';
import axios from 'axios'; // new
...
```

There are two remaining steps. First, we'll use `axios` for our `GET` request. We can make a dedicated `getTodos` function for this purpose.

Second, we want to make sure that this API call is issued at the correct time during the React lifecycle. HTTP requests should be made using componentDidMount[62] so we will call `getTodos` there.

We can also delete the mock `list` since it is no longer needed. Our complete `App.js` file will now look as follows.

Code

```
// src/App.js
import React, { Component } from 'react';
import axios from 'axios'; // new

class App extends Component {
  state = {
    todos: []
  };

  // new
  componentDidMount() {
    this.getTodos();
  }

  // new
  getTodos() {
    axios
      .get('http://127.0.0.1:8000/api/')
      .then(res => {
        this.setState({ todos: res.data });
      })
      .catch(err => {
        console.log(err);
```

[62] https://reactjs.org/docs/state-and-lifecycle.html

```
      });
  }

  render() {
    return (
      <div>
        {this.state.todos.map(item => (
          <div key={item.id}>
            <h1>{item.title}</h1>
            <span>{item.body}</span>
          </div>
        ))}
      </div>
    );
  }
}

export default App;
```

If you look again at `http://localhost:3000/` the page is the same even though we no longer have hardcoded data. It all comes from our API endpoint and request now.

API Data

And that is how it's done with React!

Conclusion

We have now connected our Django back-end API to a React front-end. Even better, we have the option to update our front-end in the future or swap it out entirely as project requirements change.

This is why adopting an API-first approach is a great way to "future-proof" your website. It may take a little more work upfront, but it provides much more flexibility. In later chapters we will enhance our APIs so they support multiple HTTP verbs such as `POST` (adding new todos), `PUT` (updating existing todos), and `DELETE` (removing todos).

In the next chapter we will start building out a robust *Blog API* that supports full CRUD (Create-Read-Update-Delete) functionality and later on add user authentication to it so users can log in, log out, and sign up for accounts via our API.

Chapter 5: Blog API

```
    """
Luồng công việc : pip => initial setup => post/model => migrate => esit
post/admin
=> config/setting => config/url => post.url => post/serialize => post/views
=>
    """
```

Dự án tiếp theo của chúng tôi là một API *Blog* sử dụng bộ đầy đủ các tính năng của Django REST Framework. Nó sẽ có người dùng, quyền và cho phép chức năng CRUD (Tạo-Đọc-Cập nhật-Xóa) đầy đủ. Chúng ta cũng sẽ khám phá các bộ chế độ xem, bộ định tuyến và tài liệu.

Sự khác biệt chính là các điểm cuối API của chúng tôi sẽ hỗ trợ CRUD ngay từ đầu, như chúng ta sẽ thấy, Django REST Framework làm cho khá liền mạch để thực hiện.

Initial Set Up

Command Line

```
$ cd ~/Desktop && cd code
$ mkdir blogapi && cd blogapi
$ pipenv install django~=3.1.0
$ pipenv shell
(blogapi) $ django-admin startproject config .
(blogapi) $ python manage.py startapp posts
```

Code

```
# config/settings.py
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
```

```
    # Local
    'posts', # new
]
```

Command Line

```
(blogapi) $ python manage.py migrate
```

Model

Mô hình cơ sở dữ liệu của chúng tôi sẽ có năm trường: tác giả, tiêu đề, nội dung, created_at và updated_at. Chúng tôi có thể sử dụng mô hình Người dùng tích hợp của Django vì tác giả miễn là chúng tôi nhập nó vào dòng thứ hai từ trên xuống.

Code

```
# posts/models.py
from django.db import models
from django.contrib.auth.models import User

class Post(models.Model):
    author = models.ForeignKey(User, on_delete=models.CASCADE)
    title = models.CharField(max_length=50)
    body = models.TextField()
    created_at = models.DateTimeField(auto_now_add=True)
    updated_at = models.DateTimeField(auto_now=True)

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

Command Line

```
(blogapi) $ python manage.py makemigrations posts
Migrations for 'posts':
  posts/migrations/0001_initial.py
    - Create model Post
(blogapi) $ python manage.py migrate
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, posts, sessions
Running migrations:
  Applying posts.0001_initial... OK
```

Code

```
# posts/admin.py
from django.contrib import admin
from .models import Post
```

```
admin.site.register(Post)
```

Command Line

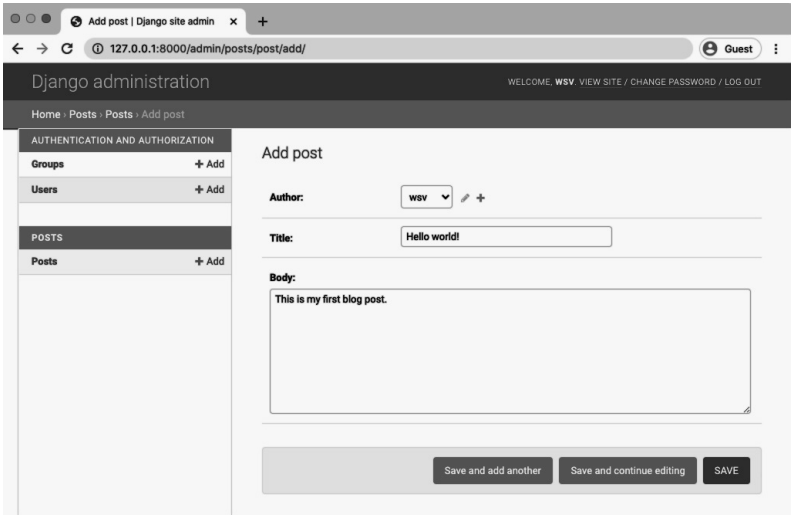
```
(blogapi) $ python manage.py createsuperuser
```

Now we can start up the local web server.

Command Line

```
(blogapi) $ python manage.py runserver
```

Navigate to `http://127.0.0.1:8000/admin/`



Admin add blog post

Tests

Let's write a basic test for our `Post` model. We want to ensure a logged-in user can create a blog post with a title and body.

Code

```
# posts/tests.py
from django.test import TestCase
from django.contrib.auth.models import User
from .models import Post


class BlogTests(TestCase):
```

```
    @classmethod
    def setUpTestData(cls):
        # Create a user
        testuser1 = User.objects.create_user(
            username='testuser1', password='abc123')
        testuser1.save()

        # Create a blog post
        test_post = Post.objects.create(
            author=testuser1, title='Blog title', body='Body content...')
        test_post.save()

    def  test_blog_content(self):
        post = Post.objects.get(id=1)
        author = f'{post.author}'
        title = f'{post.title}'
        body = f'{post.body}'
        self.assertEqual(author, 'testuser1')
        self.assertEqual(title, 'Blog title')
        self.assertEqual(body, 'Body content...')
```

To confirm that our tests are working quit the local server `Control+c`. Then run our tests.

Command Line

```
(blogapi) $ python manage.py test
```

Command Line

```
(blogapi) $ python manage.py test
Creating test database for alias 'default'...
System check identified no issues (0 silenced).
.
----------------------------------------------------------------------
Ran 1 test in 0.105s

OK
Destroying test database for alias 'default'...
```

Django REST Framework

Như chúng ta đã thấy trước đây, Django REST Framework đảm nhận công việc nặng nhọc trong việc chuyển đổi các mô hình cơ sở dữ liệu của chúng ta thành một API RESTful. Có ba bước chính cho quá trình này:

- `urls.py` cho các tuyến URL
- `serializers.py` để chuyển đổi dữ liệu thành JSON
- `views.py` để áp dụng logic cho mỗi điểm cuối API

Command Line

```
(blogapi) $ pipenv install djangorestframework~=3.11.0
```

Code

```
# config/settings.py
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',

    # 3rd-party apps
    'rest_framework', # new

    # Local
    'posts.apps.PostsConfig',
]

# new
REST_FRAMEWORK = {
    'DEFAULT_PERMISSION_CLASSES': [
        'rest_framework.permissions.AllowAny',
    ]
}
```

URLs

Code

```
# config/urls.py
from django.contrib import admin
from django.urls import include, path # new

urlpatterns = [
    path('admin/', admin.site.urls),
    path('api/v1/', include('posts.urls')), # new
]
```

Thực hành tốt là luôn tạo phiên bản API của bạn — v1 /, v2 /, v.v. — vì khi bạn thực hiện một thay đổi lớn, có thể có một số thời gian trễ trước khi những người tiêu dùng API khác nhau cũng có thể cập nhật. Bằng cách đó, bạn có thể hỗ trợ v1 của API trong một khoảng thời gian đồng thời khởi chạy phiên bản 2 mới, được cập nhật và tránh phá vỡ các ứng dụng khác dựa vào back-end API của bạn.

Command Line

```
(blogapi) $ touch posts/urls.py
```

Code

```
# posts/urls.py
from django.urls import path
from .views import PostList, PostDetail

urlpatterns = [
    path('<int:pk>/', PostDetail.as_view()),
    path('', PostList.as_view()),
]
```

Serializers

Command Line

```
(blogapi) $ touch posts/serializers.py
```

Code

```
# posts/serializers.py
from rest_framework import serializers
from .models import Post


class PostSerializer(serializers.ModelSerializer):

    class Meta:
        fields = ('id', 'author', 'title', 'body', 'created_at',)
        model = Post
```

Views

Bước cuối cùng là tạo ra quan điểm của chúng tôi. Django REST Framework có một số quan điểm

chung rất hữu ích. Chúng tôi đã sử dụng ListAPIView[63] trong cả API *Library* và *Todos* để tạo bộ sưu tập điểm cuối chỉ đọc, về cơ bản là danh sách tất cả các phiên bản mô hình. Trong API *Todos*, chúng tôi cũng đã sử dụng RetrieveAPIView[64] cho một điểm cuối chỉ đọc, tương tự như chế độ xem chi tiết trong Django truyền thống.

Đối với API *Blog* của chúng tôi, chúng tôi muốn liệt kê tất cả các bài đăng trên blog có sẵn dưới dạng điểm cuối đọc-ghi, có nghĩa là sử dụng ListCreateAPIView [65], tương tự như ListAPIView mà chúng tôi đã sử dụng trước đây nhưng cho phép ghi. Chúng tôi cũng muốn làm cho các bài đăng trên blog riêng lẻ có sẵn để đọc, cập nhật hoặc xóa. Và chắc chắn, có một chế độ xem Khung Django REST chung được tích hợp sẵn chỉ cho mục đích này: RetrieveUpdateDestroyAPIView[66]. Đó là những gì chúng tôi sẽ sử dụng ở đây.

Code

```
# posts/views.py
from rest_framework import generics
from .models import Post
from .serializers import PostSerializer


class PostList(generics.ListCreateAPIView):
    queryset = Post.objects.all()
    serializer_class = PostSerializer


class PostDetail(generics.RetrieveUpdateDestroyAPIView):
    queryset = Post.objects.all()
    serializer_class = PostSerializer
```

Browsable API

Command Line

```
(blogapi) $ python manage.py runserver
```

Then go to `http://127.0.0.1:8000/api/v1/` to see the Post List endpoint.

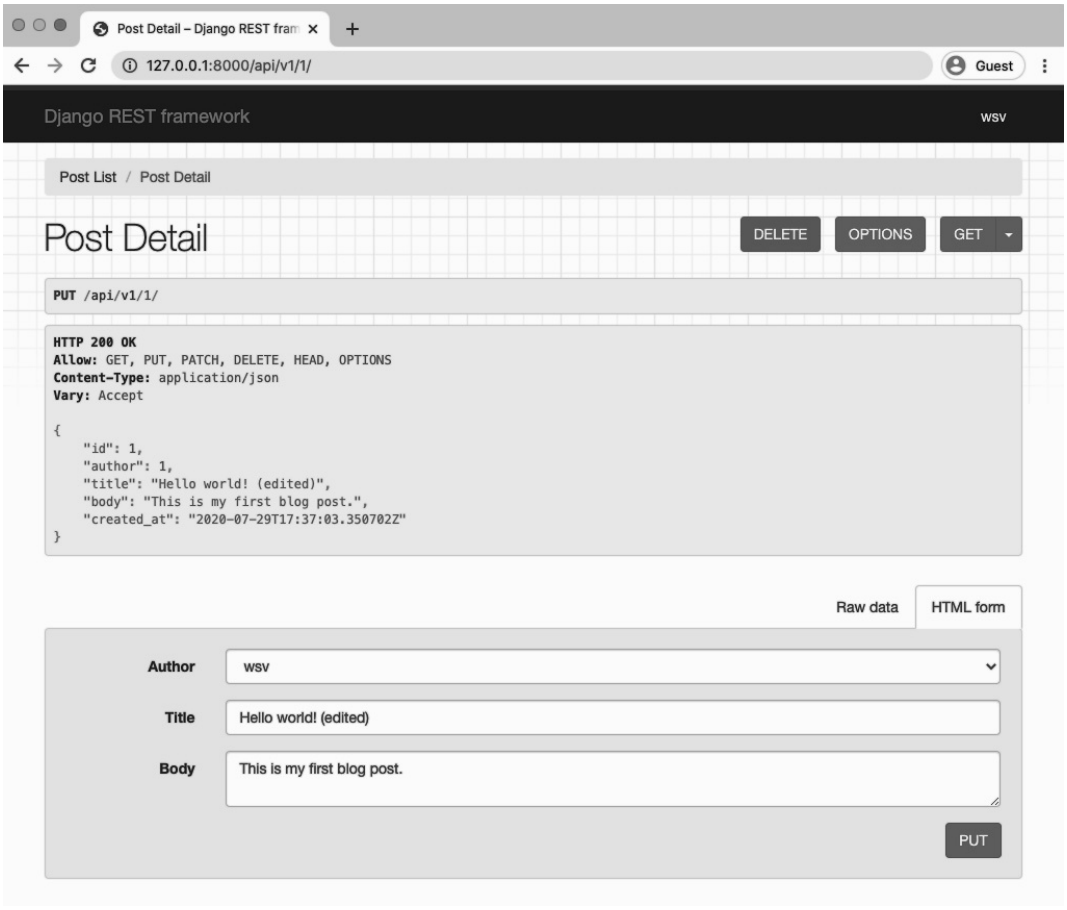
API Post List

Go to `http://127.0.0.1:8000/api/v1/1/`.

API Post Detail

Bạn có thể thấy trong tiêu đề rằng GET, PUT, PATCH và DELETE được hỗ trợ nhưng không hỗ trợ POST. Và trên thực tế, bạn có thể sử dụng biểu mẫu HTML bên dưới để thực hiện các thay đổi hoặc thậm chí sử dụng nút "XÓA" màu đỏ để xóa phiên bản.

Hãy thử mọi thứ. Cập nhật tiêu đề của chúng tôi với văn bản bổ sung (đã chỉnh sửa) ở cuối. Sau đó nhấp vào nút "ĐẶT".



API Post Detail edited

Go back to the Post List view by clicking on the link for it at the top of the page or navigating directly to `http://127.0.0.1:8000/api/v1/` and you can see the updated text there as well.

Conclusion

API *Blog* của chúng tôi hoàn toàn hoạt động tại thời điểm này. Tuy nhiên, có một vấn đề lớn: bất kỳ ai cũng có thể cập nhật hoặc xóa một bài đăng trên blog hiện có! Nói cách khác, chúng tôi không có bất kỳ quyền nào tại chỗ. Trong chương tiếp theo, chúng ta sẽ tìm hiểu cách áp dụng các quyền để bảo vệ API của chúng ta.

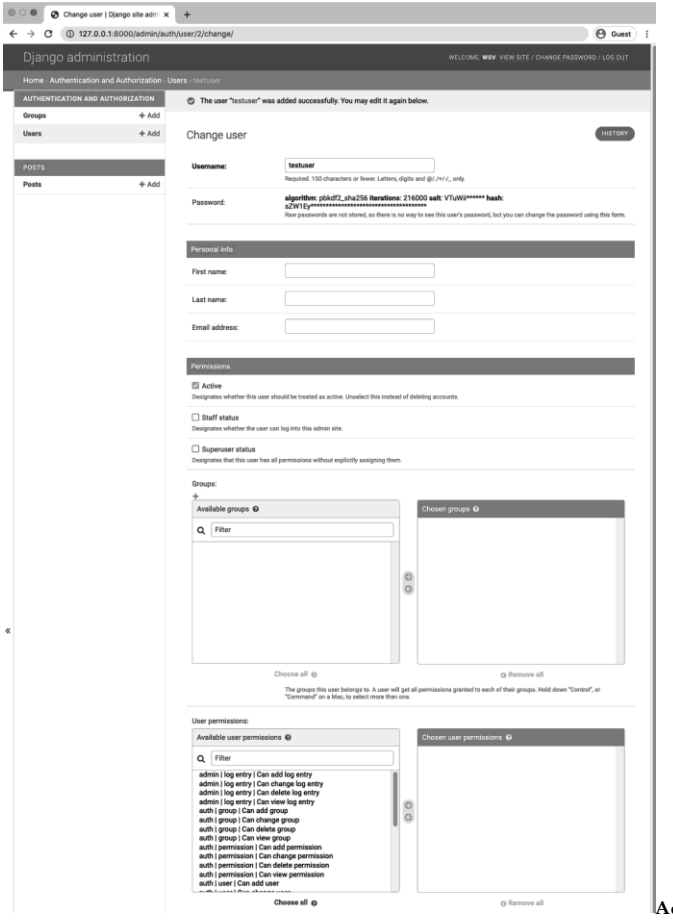
Chapter 6: Permissions

Django REST Framework đi kèm với một số cài đặt quyền có sẵn mà chúng tôi có thể sử dụng để bảo mật API của mình. Chúng có thể được áp dụng ở cấp độ dự án, cấp độ xem hoặc ở bất kỳ cấp độ mô hình riêng lẻ nào.

Trong chương này, chúng tôi sẽ thêm một người dùng mới và thử nghiệm với nhiều cài đặt quyền. Sau đó , chúng tôi sẽ tạo quyền tùy chỉnh của riêng mình để chỉ tác giả của bài đăng trên blog mới có khả năng cập nhật hoặc xóa bài đăng đó.

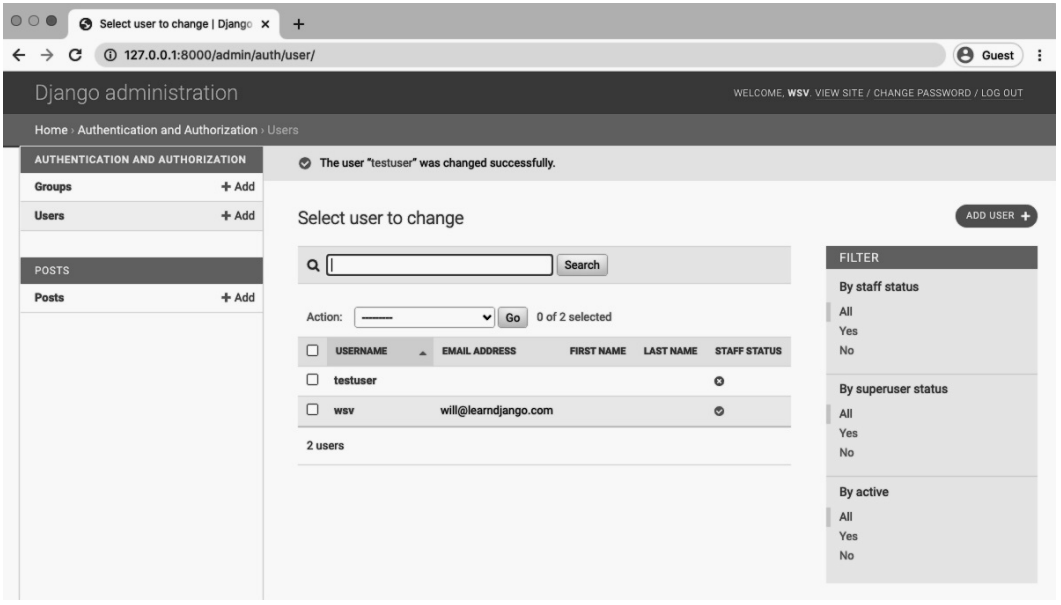
Create a new user

Navigate to the admin at `http://127.0.0.1:8000/admin/`. Then click on "+ Add" next to `Users`.



Admin User Change

Scroll down to the bottom of this page and click the “Save” button. It will redirect back to the main Users page at http://127.0.0.1:8000/admin/auth/user/[67].



Admin All Users

We can see our two users are listed.

Add log in to the browsable API

Trong tệp urls.py cấp dự án, hãy thêm tuyến URL mới bao gồm rest_framework.url. Hơi khó hiểu, tuyến đường thực tế được chỉ định có thể là bất cứ thứ gì chúng ta muốn; Điều quan trọng là rest_framework.urls được bao gồm *ở đâu đó*. Chúng ta sẽ sử dụng route api-auth vì nó phù hợp với

[67] http://127.0.0.1:8000/admin/auth/user/

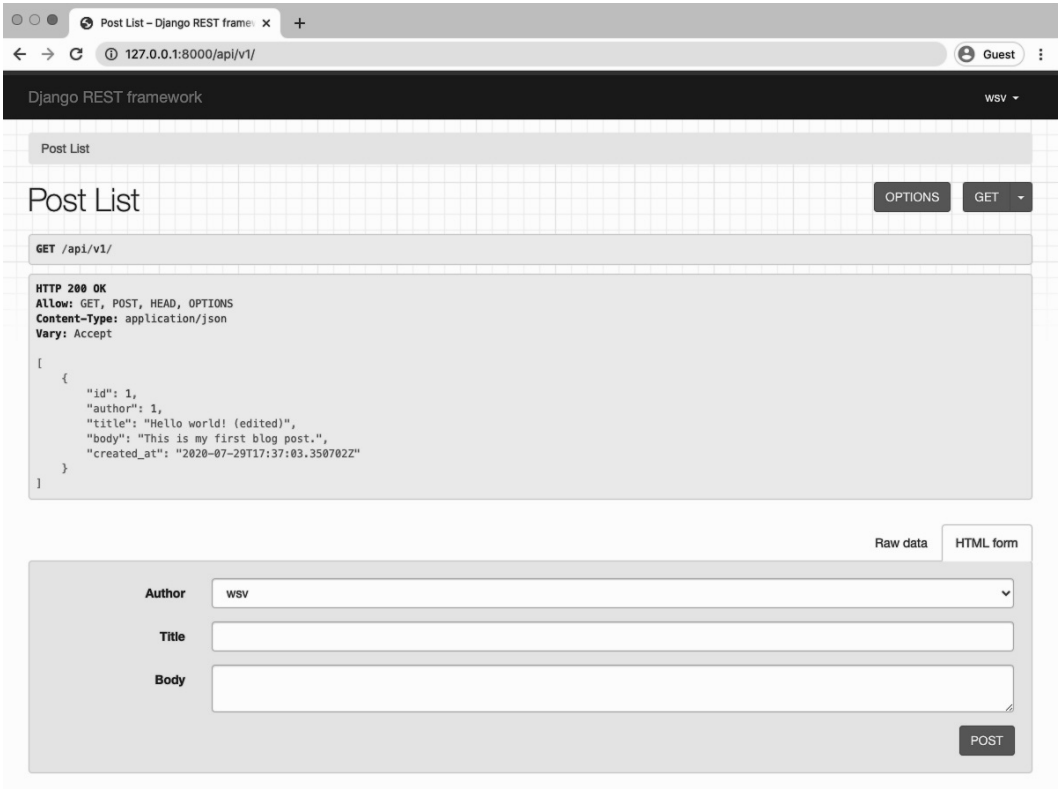
tài liệu chính thức, nhưng chúng tôi có thể dễ dàng sử dụng bất cứ thứ gì chúng tôi muốn và mọi thứ sẽ hoạt động giống nhau.

Code

```
# config/urls.py
from django.contrib import admin
from django.urls import include, path

urlpatterns = [
    path('admin/', admin.site.urls),
    path('api/v1/', include('posts.urls')),
```

```
    path('api-auth/', include('rest_framework.urls')), # new
]
```

Bây giờ điều hướng đến API có thể duyệt của chúng tôi tại http://127.0.0.1:8000/api/v1/. Có một thay đổi tinh tế: bên cạnh tên người dùng ở góc trên bên phải là một mũi tên hướng xuống một chút.



API Log In

Vì chúng tôi đã đăng nhập bằng tài khoản superuser của mình tại thời điểm này — wsv cho tôi — tên đó xuất hiện. Nhấp vào liên kết và menu thả xuống với "Đăng xuất" xuất hiện. Nhấp vào nó.

Liên kết trên bên phải hiện thay đổi thành "Đăng nhập". Vì vậy, hãy nhấp vào đó. Chúng tôi được chuyển hướng đến trang đăng nhập của Django REST Framework. Sử dụng tài khoản testuser của chúng tôi tại đây. Cuối cùng nó sẽ chuyển hướng chúng ta trở lại trang API chính nơi testuser hiện diện ở góc trên bên phải.

AllowAny

Và trên trang chi tiết http://127.0.0.1:8000/api/v1/1/ thông tin cũng hiển thị và bất kỳ người dùng ngẫu nhiên nào cũng có thể cập nhật hoặc xóa một bài đăng trên blog hiện có. Không tốt.

Lý do chúng ta vẫn có thể thấy điểm cuối Danh sách bài đăng và cả điểm cuối Danh sách chi tiết là vì chúng ta đã đặt trước các quyền cấp dự án trên dự án của mình thành AllowAny trong tệp config / settings.py của chúng ta . Như một lời nhắc nhở ngắn gọn, nó trông như thế này:

Code

```
# config/settings.py
REST_FRAMEWORK = {
    'DEFAULT_PERMISSION_CLASSES': [
        'rest_framework.permissions.AllowAny',
    ]
}
```

View-Level Permissions

Những gì chúng tôi muốn bây giờ là hạn chế quyền truy cập API đối với người dùng đã xác thực. Có nhiều nơi chúng tôi có thể làm điều này—project-level, view-level, or object-level—Nhưng vì chúng tôi chỉ có hai chế độ xem vào lúc này, hãy bắt đầu từ đó và thêm quyền cho từng chế độ xem .

Code

```
# posts/views.py
from rest_framework import generics, permissions  # new
from .models import Post
from .serializers import PostSerializer


class  PostList(generics.ListCreateAPIView):
    permission_classes = (permissions.IsAuthenticated,)  # new
    queryset = Post.objects.all()
    serializer_class = PostSerializer


class PostDetail(generics.RetrieveUpdateDestroyAPIView):
```

```
    permission_classes = (permissions.IsAuthenticated,)  # new
    queryset = Post.objects.all()
    serializer_class = PostSerializer
```

That's all we need. Refresh the browsable API at `http://127.0.0.1:8000/api/v1/`. Look what happened!

API Post List Logged Out

Chúng tôi không còn thấy trang Danh sách bài đăng của chúng tôi nữa. Thay vào đó, chúng tôi được chào đón bằng một người không thân thiện`HTTP 403 Forbidden` status code vì chúng tôi chưa đăng nhập. Và không có biểu mẫu nào trong API có thể duyệt để chỉnh sửa dữ liệu vì chúng tôi không có quyền.

If you use the URL for Post Detail `http://127.0.0.1:8000/api/v1/1/` you will see a similar message and also no available forms for edits.

API Detail Logged Out

. Việc thêm một permission_classes chuyên dụng vào mỗi chế độ xem có vẻ lặp đi lặp lại nếu chúng ta

muốn đặt cùng một cài đặt quyền trên toàn bộ API của mình.

Sẽ không tốt hơn nếu thay đổi quyền của chúng tôi một lần, lý tưởng nhất là ở cấp dự án, thay vì làm điều đó cho mỗi và mọi chế độ xem?

Project-Level Permissions

May mắn thay , Django REST Framework đi kèm với một số cài đặt quyền cấp dự án tích hợp sẵn mà chúng tôi có thể sử dụng, bao gồm:

1. AllowAny[68] -Bất kỳ người dùng nào, được xác thực hay không, đều có toàn quyền truy cập
   - IsAuthenticated[69] - Chỉ những người dùng đã đăng ký, được xác thực mới có quyền truy cập
1. IsAdminUser[70] - Chỉ quản trị viên / siêu người dùng mới có quyền truy cập
   - IsAuthenticatedOrReadOnly[71] - Người dùng trái phép có thể xem bất kỳ trang nào, nhưng chỉ những người dùng authenti- cated mới có đặc quyền viết, chỉnh sửa hoặc xóa

Việc triển khai bất kỳ cài đặt nào trong số bốn cài đặt này yêu cầu cập nhật DEFAULT_PERMISSION_CLASSES cài đặt và làm mới trình duyệt web của chúng tôi. Đó là nó!

Hãy chuyển sang IsAuthenticated để chỉ người dùng được xác thực hoặc đăng nhập mới có thể xem API. Cập nhật tệp config/settings.py như sau:

Code

```
# config/settings.py
REST_FRAMEWORK = {
    'DEFAULT_PERMISSION_CLASSES': [
        'rest_framework.permissions.IsAuthenticated', # new
    ]
}
```

Bây giờ quay lại tệp bài đăng / lượt xem.py và xóa các thay đổi quyền mà chúng tôi vừa thực hiện.

Code

```
# posts/views.py
from rest_framework import generics
from .models import Post
from .serializers import PostSerializer


class PostList(generics.ListCreateAPIView):
    queryset = Post.objects.all()
    serializer_class = PostSerializer
```

```
class PostDetail(generics.RetrieveUpdateDestroyAPIView):
    queryset = Post.objects.all()
    serializer_class = PostSerializer
```

Custom permissions

Chúng tôi chỉ muốn tác giả của một bài đăng blog cụ thể có thể chỉnh sửa hoặc xóa nó; nếu không, bài đăng trên blog sẽ ở chế độ chỉ đọc. Vì vậy, tài khoản superuser nên có quyền truy cập CRUD đầy đủ vào phiên bản blog cá nhân, nhưng người dùng thử nghiệm thông thường thì không.

`Control+c` and create a new `permissions.py` file in our `posts` app.

Command Line

```
(blogapi) $ touch posts/permissions.py
```

Trong nội bộ, Django REST Framework dựa trên một lớp BasePermission mà từ đó tất cả các lớp cho mỗi nhiệm vụ khác kế thừa. Điều đó có nghĩa là các cài đặt quyền tích hợp như AllowAny, IsAuthenticated và các cài đặt khác sẽ mở rộng nó. Đây là mã nguồn thực tế có sẵn trên Github[72]:

Code

```
class BasePermission(object):
    """
    A base class from which all permission classes should inherit.
    """

    def has_permission(self, request, view):
        """
        Return `True` if permission is granted, `False` otherwise.
        """
        return True

    def has_object_permission(self, request, view, obj):
        """
        Return `True` if permission is granted, `False` otherwise.
        """
        return True
```

Để tạo quyền tùy chỉnh của riêng mình, chúng tôi sẽ ghi đè phương thức has_object_permission. Cụ thể, chúng tôi muốn cho phép chỉ đọc cho tất cả các yêu cầu nhưng đối với bất kỳ yêu cầu ghi nào, chẳng hạn như chỉnh sửa hoặc xóa, tác giả phải giống với người dùng đã đăng nhập hiện tại.

Code

```
# posts/permissions.py
from rest_framework import permissions


class IsAuthorOrReadOnly(permissions.BasePermission):

    def has_object_permission(self, request, view, obj):
        # Read-only permissions are allowed for any request
        if request.method in permissions.SAFE_METHODS:
            return True

        # Write permissions are only allowed to the author of a post
        return obj.author == request.user
```

Chúng ta import quyền ở trên cùng và sau đó tạo một custom class IsAuthorOrReadOnly mở rộng BasePermission. Sau đó, chúng tôi ghi đè lên has_object_permission. Nếu một yêu cầu chứa các động từ HTTP có trong SAFE_METHODS – một bộ chứa GET, OPTIONS và HEAD – thì đó là yêu cầu chỉ đọc và quyền được cấp.

Nếu không, yêu cầu là ghi một số loại, có nghĩa là cập nhật tài nguyên API để tạo, xóa hoặc chỉnh sửa chức năng. Trong trường hợp đó, chúng ta kiểm tra xem tác giả của đối tượng được đề cập hay không, đó là bài đăng trên blog của chúng ta obj.author có khớp với người dùng đưa ra request request.user hay không.

Quay lại tệp views.py, chúng ta nên nhập IsAuthorOrReadOnly và sau đó chúng ta có thể thêm các lớp permission_- cho PostDetail.

Code

```
# posts/views.py
from rest_framework import generics
from .models import Post
from .permissions import IsAuthorOrReadOnly # new
from .serializers import PostSerializer


class PostList(generics.ListCreateAPIView):
    queryset = Post.objects.all()
    serializer_class = PostSerializer


class PostDetail(generics.RetrieveUpdateDestroyAPIView):
```

```
    permission_classes = (IsAuthorOrReadOnly,)  # new
    queryset = Post.objects.all()
    serializer_class = PostSerializer
```

Và chúng tôi đã hoàn tất. Hãy kiểm tra mọi thứ. Điều hướng đến trang Chi tiết bài đăng, nằm ở http://127.0.0.1:8000/api/v1/1/. Đảm bảo rằng bạn đã đăng nhập bằng tài khoản siêu người dùng của mình, tác giả của bài đăng. Tên người dùng sẽ hiển thị ở góc trên bên phải của trang.

API Detail Superuser

Tuy nhiên, nếu bạn đăng xuất và sau đó đăng nhập bằng tài khoản testuser, trang sẽ thay đổi.

Chúng tôi **có thể** xem trang này vì các quyền chỉ đọc được cho phép. Tuy nhiên chúng tôi **không thể** làm cho bất kỳ

Các yêu cầu PUT hoặc DELETE do lớp quyền IsAuthorOrReadOnly tùy chỉnh của chúng tôi.

Note that the generic views will only check the object-level permissions for views that retrieve a single model instance. If you require object-level filtering of list views–for a collection of instances–you'll need to filter by overriding the initial queryset[73].

Conclusion

Setting proper permissions is a very important part of any API. As a general strategy, it is a good idea to set a strict project-level permissions policy such that only authenticated users can view the API. Then make view-level or custom permissions more accessible as needed on specific API endpoints.

[73] http://www.django-rest-framework.org/api-guide/filtering/#overriding-the-initial-queryset

Chapter 7: User Authentication

Trong chương trước, chúng tôi đã cập nhật các quyền API của mình, còn được gọi là **ủy quyền**. Trong chương này, chúng tôi sẽ triển khai **xác thực**, đây là quá trình mà người dùng có thể đăng ký tài khoản mới, đăng nhập bằng tài khoản đó và đăng xuất.

Giải pháp là chuyển một mã định danh duy nhất với mỗi yêu cầu HTTP. Thật khó hiểu, không có cách tiếp cận nào được thống nhất trên toàn cầu đối với hình thức của mã định danh này và nó có thể có nhiều dạng. Django REST Framework đi kèm với bốn tùy chọn xác thực tích hợp khác nhau[74]: basic, session, token, and default. Và có nhiều gói bên thứ ba khác cung cấp các tính năng bổ sung như Mã thông báo web JSON (JWT).

Trong chương này, chúng tôi sẽ khám phá kỹ lưỡng cách xác thực API hoạt động, xem xét ưu và nhược điểm của từng cách tiếp cận, sau đó đưa ra lựa chọn sáng suốt cho API *Blog* của chúng tôi. Cuối cùng, chúng tôi sẽ tạo các điểm cuối API để đăng ký, đăng nhập và đăng xuất.

Basic Authentication

Hình thức xác thực HTTP phổ biến nhất được gọi là Xác thực "Cơ bản"[75]. Khi máy khách đưa ra yêu cầu HTTP, nó buộc phải gửi thông tin xác thực đã được phê duyệt trước khi quyền truy cập được cấp.

Luồng yêu cầu /phản hồi hoàn chỉnh trông như thế này:Client tạo một yêu cầu HTTP

1. Server responds with an HTTP response containing a `401 (Unauthorized)` status code and `WWW-Authenticate` HTTP header with details on *how* to authorize
2. Client sends credentials back via the Authorization[76] HTTP header
3. Server checks credentials and responds with either `200 OK` or `403 Forbidden` status code

Sau khi được phê duyệt, máy khách sẽ gửi tất cả các yêu cầu trong tương lai với thông tin đăng nhập tiêu đề HTTP Ủy quyền. Chúng ta cũng có thể hình dung cuộc trao đổi này như sau:

Diagram

```
Client                                                  Server
------                                                  ------
```

```
--------------------------------------->
GET / HTTP/1.1

                                        <---------------------------------------
                                                    HTTP/1.1 401 Unauthorized
                                                    WWW-Authenticate: Basic

--------------------------------------->
GET / HTTP/1.1
Authorization: Basic d3N2OnBhc3N3b3JkMTIz

                                        <---------------------------------------
                                                                HTTP/1.1 200 OK
```

Lưu ý rằng thông tin xác thực ủy quyền được gửi là thông tin đăng nhập không được mã hóa base64 encoded[77] version of `<username>:<password>`. Vì vậy, trong trường hợp của tôi, đây là `wsv:password123` which with base64 encoding is `d3N2OnBhc3N3b3JkMTIz`.

Ưu điểm chính của phương pháp này là sự đơn giản của nó. Nhưng có một số nhược điểm lớn. Đầu tiên, trên *mỗi yêu cầu*, máy chủ phải tra cứu và xác minh tên người dùng và mật khẩu, điều này không hiệu quả. Sẽ tốt hơn nếu bạn thực hiện tra cứu một lần và sau đó chuyển một mã thông báo nào đó cho biết, người dùng này được chấp thuận. Thứ hai, thông tin đăng nhập của người dùng đang được chuyển dưới dạng văn bản rõ ràng — hoàn toàn không được mã hóa — qua internet. Điều này là vô cùng bất an. Bất kỳ lưu lượng truy cập internet nào không

[76] https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Authorization
[77] https://en.wikipedia.org/wiki/Base64

Mã hóa có thể dễ dàng bị bắt và tái sử dụng. Do đó, xác thực cơ bản **chỉ** nên được sử dụng thông qua HTTPS[78], phiên bản an toàn của HTTP.

Session Authentication

Các trang web nguyên khối, giống như Django truyền thống, từ lâu đã sử dụng một sơ đồ xác thực thay thế là sự kết hợp giữa phiên và cookie. Ở cấp độ cao, máy khách xác thực bằng thông tin đăng nhập của nó (tên người dùng / mật khẩu) và sau đó nhận được ID *phiên từ* máy chủ được lưu trữ dưới dạng cookie. ID phiên này sau đó được chuyển trong tiêu đề của mọi yêu cầu HTTP trong tương lai.

Khi ID phiên được thông qua, máy chủ sử dụng nó để tra cứu một đối tượng phiên chứa tất cả thông tin có sẵn cho một người dùng nhất định, bao gồm cả thông tin đăng nhập.

Cách tiếp cận này là trạng thái vì một bản ghi phải được lưu giữ và duy trì trên cả máy chủ (đối tượng phiên) và máy khách (ID phiên).

Let's review the basic flow:

1. A user enters their log in credentials (typically username/password)
2. The server verifies the credentials are correct and generates a session object that is then stored in the database
3. The server sends the client a session ID—not the session object itself—which is stored as a cookie on the browser
4. On all future requests the session ID is included as an HTTP header and, if verified by the database, the request proceeds
5. Once a user logs out of an application, the session ID is destroyed by both the client and server
6. If the user later logs in again, a new session ID is generated and stored as a cookie on the client

Cài đặt mặc định trong Django REST Framework thực sự là sự kết hợp giữa Xác thực cơ bản và Xác thực phiên. Hệ thống xác thực dựa trên phiên truyền thống của Django được sử dụng và ID phiên được chuyển trong tiêu đề HTTP trên mỗi yêu cầu thông qua Xác thực cơ bản.

[78] https://en.wikipedia.org/wiki/HTTPS

Ưu điểm của phương pháp này là an toàn hơn vì thông tin đăng nhập của người dùng chỉ được gửi một lần, không phải trên mọi chu kỳ yêu cầu / phản hồi như trong Xác thực cơ bản. Nó cũng hiệu quả hơn vì máy chủ không phải xác minh thông tin đăng nhập của người dùng mỗi lần, nó chỉ khớp ID phiên với đối tượng phiên được tra cứu nhanh chóng.

Tuy nhiên, có một số nhược điểm. Đầu tiên, ID phiên chỉ hợp lệ trong trình duyệt nơi đăng nhập được thực hiện; nó sẽ không hoạt động trên nhiều miền. Đây là một vấn đề hiển nhiên khi một API cần hỗ trợ nhiều giao diện người dùng như trang web và ứng dụng dành cho thiết bị di động. Thứ hai, đối tượng phiên phải được cập nhật, điều này có thể là thách thức trong các trang web lớn có nhiều máy chủ. Làm thế nào để bạn duy trì độ chính xác của một đối tượng phiên trên mỗi máy chủ? Và thứ ba, cookie được gửi đi cho mọi yêu cầu, ngay cả những yêu cầu không yêu cầu xác thực, điều này không hiệu quả.

Do đó, thường không nên sử dụng sơ đồ xác thực dựa trên phiên cho bất kỳ API nào sẽ có nhiều giao diện người dùng.

Token Authentication

Cách tiếp cận chính thứ ba – và cách chúng tôi sẽ triển khai trong API *Blog* của mình – là sử dụng xác thực mã thông báo. Đây là cách tiếp cận phổ biến nhất trong những năm gần đây do sự gia tăng của các ứng dụng trang đơn.

Xác thực dựa trên mã thông báo là **không có trạng thái**: khi máy khách gửi thông tin đăng nhập người dùng ban đầu đến máy chủ, một mã thông báo duy nhất sẽ được tạo và sau đó được máy khách lưu trữ dưới dạng cookie hoặc trong local storage[79]. Mã thông báo này sau đó được chuyển trong tiêu đề của mỗi yêu cầu HTTP đến và máy chủ sử dụng nó để xác minh rằng người dùng được xác thực. Bản thân máy chủ không lưu giữ hồ sơ của người dùng, chỉ là mã thông báo có hợp lệ hay không.

[79] https://developer.mozilla.org/en-US/docs/Web/API/Window/localStorage

> Tokens stored in both cookies and localStorage are vulnerable to XSS attacks. The current best practice is to store tokens in a cookie with the `httpOnly` and `Secure` cookie flags.

Hãy xem xét một phiên bản đơn giản của các thông báo HTTP thực tế trong này challenge/response flow. Lưu ý rằng tiêu đề HTTP WWW-Authenticate chỉ định việc sử dụng Mã thông báo được sử dụng trong yêu cầu tiêu đề Ủy quyền phản hồi.

Diagram

```
Client                                                              Server
------                                                              ------

--------------------------------------->
GET / HTTP/1.1

                                      <---------------------------------------
                                                  HTTP/1.1 401 Unauthorized
                                                  WWW-Authenticate: Token

--------------------------------------->
GET / HTTP/1.1
Authorization: Token 401f7ac837da42b97f613d789819ff93537bee6a

                                      <---------------------------------------
                                                        HTTP/1.1 200 OK
```

Có nhiều lợi ích cho phương pháp này. Vì mã thông báo được lưu trữ trên máy khách, việc thay đổi quy mô máy chủ để duy trì các đối tượng phiên cập nhật không còn là vấn đề nữa. Và mã thông báo có thể được chia sẻ giữa nhiều giao diện người dùng: cùng một mã thông báo có thể đại diện cho người dùng trên trang web và cùng một người dùng trên ứng dụng dành cho thiết bị di động. Không thể chia sẻ cùng một ID phiên giữa các giao diện người dùng khác nhau, một hạn chế lớn.

Một nhược điểm tiềm ẩn là mã thông báo có thể phát triển khá lớn. Mã thông báo chứa tất cả thông tin người dùng, không chỉ là id như với id phiên / đối tượng phiên được thiết lập. Vì mã thông báo được gửi theo mọi yêu cầu, việc quản lý kích thước của nó có thể trở thành một vấn đề về hiệu suất.

Chính xác *cách* mã thông báo được triển khai cũng có thể khác nhau đáng kể. Khung Django REST '

TokenAuthentication[80] tích hợp có chủ ý khá cơ bản. Do đó, nó không hỗ trợ đặt mã thông báo hết hạn, đây là một cải tiến bảo mật có thể được thêm vào. Nó cũng chỉ tạo một mã thông báo cho mỗi người dùng , vì vậy một người dùng trên một trang web và sau đó là một ứng dụng dành cho thiết bị di động sẽ sử dụng cùng một mã thông báo. Vì thông tin về người dùng được lưu trữ cục bộ, điều này có thể gây ra sự cố với việc duy trì và cập nhật hai bộ thông tin khách hàng.

JSON Web Tokens (JWTs) là một phiên bản mới, nâng cao của mã thông báo có thể được thêm vào Django REST Framework thông qua một số gói của bên thứ ba. JWT có một số lợi ích bao gồm khả năng tạo mã thông báo khách hàng duy nhất và hết hạn mã thông báo. Chúng có thể được tạo trên máy chủ hoặc với dịch vụ của bên thứ ba như Auth0[81]. Và JWT có thể được mã hóa, giúp chúng an toàn hơn khi gửi qua các kết nối HTTP không an toàn.

Cuối cùng, đặt cược an toàn nhất cho hầu hết các API web là sử dụng sơ đồ xác thực dựa trên mã thông báo. JWT là một bổ sung tốt đẹp, hiện đại mặc dù chúng yêu cầu cấu hình bổ sung. Do đó, trong cuốn sách này, chúng tôi sẽ sử dụng TokenAuthentication tích hợp sẵn.

Default Authentication

Bước đầu tiên là định cấu hình cài đặt xác thực mới của chúng tôi. Django REST Framework đi kèm với một số cài đặt [82] được đặt ngầm. Ví dụ: DEFAULT_PERMISSION_CLASSES đã được đặt thành AllowAny trước khi chúng tôi cập nhật nó lên IsAuthenticated.

Các DEFAULT_AUTHENTICATION_CLASSES được đặt theo mặc định , vì vậy hãy thêm rõ ràng cả hai SessionAuthentication and `BasicAuthentication` to our `config/settings.py` file.

[80] http://www.django-rest-framework.org/api-guide/authentication/#tokenauthentication
[81] https://auth0.com/
[82] http://www.django-rest-framework.org/api-guide/settings/

Code

```
REST_FRAMEWORK = {
    'DEFAULT_PERMISSION_CLASSES': [
        'rest_framework.permissions.IsAuthenticated',
    ],
    'DEFAULT_AUTHENTICATION_CLASSES': [ # new
        'rest_framework.authentication.SessionAuthentication',
        'rest_framework.authentication.BasicAuthentication'
    ],
}
```

Tại sao sử dụng **cả hai** phương pháp? Câu trả lời là chúng phục vụ các mục đích khác nhau. Các phiên được sử dụng để cung cấp năng lượng cho API có thể duyệt và khả năng đăng nhập và đăng xuất khỏi nó. BasicAuthentication được sử dụng để chuyển ID phiên trong tiêu đề HTTP cho chính API.

Nếu bạn truy cập lại API có thể duyệt tại http://127.0.0.1:8000/api/v1/ nó sẽ hoạt động như trước đây. Về mặt kỹ thuật, không có gì thay đổi, chúng tôi vừa thực hiện các cài đặt mặc định rõ ràng.

Implementing token authentication

Bây giờ chúng ta cần cập nhật hệ thống xác thực của mình để sử dụng mã thông báo. Bước đầu tiên là cập nhật của chúng tôi

DEFAULT_AUTHENTICATION_CLASSES cài đặt để sử dụng TokenAuthentication như sau:

Code

```
# config/settings.py
REST_FRAMEWORK = {
    'DEFAULT_PERMISSION_CLASSES': [
        'rest_framework.permissions.IsAuthenticated',
    ],
    'DEFAULT_AUTHENTICATION_CLASSES': [
        'rest_framework.authentication.SessionAuthentication',
        'rest_framework.authentication.TokenAuthentication', # new
    ],
}
```

Chúng tôi giữ SessionAuthentication vì chúng tôi vẫn cần nó cho API Có thể duyệt của mình, nhưng bây giờ sử dụng mã thông báo để chuyển thông tin xác thực qua lại trong tiêu đề HTTP của chúng tôi. Chúng tôi cũng cần thêm

ứng dụng authtoken tạo ra các mã thông báo trên máy chủ. Nó đi kèm với Django REST Framework nhưng phải được thêm vào cài đặt INSTALLED_APPS của chúng tôi:

Code

```
# config/settings.py
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',

    # 3rd-party apps
    'rest_framework',
    'rest_framework.authtoken', # new

    # Local
    'posts',
]
```

Vì chúng tôi đã thực hiện các thay đổi đối với INSTALLED_APPS của mình, chúng tôi cần đồng bộ hóa cơ sở dữ liệu của mình. Dừng máy chủ bằng Control+c. Sau đó chạy lệnh sau.

Command Line

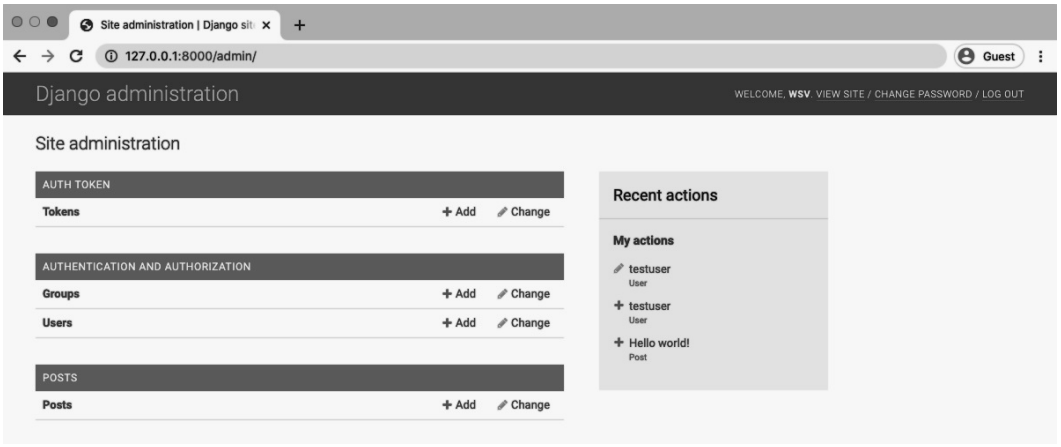
```
(blogapi) $ python manage.py migrate
Operations to perform:
  Apply all migrations: admin, auth, authtoken, contenttypes, posts, sessions
Running migrations:
  Applying authtoken.0001_initial... OK
  Applying authtoken.0002_auto_20160226_1747... OK
```

Now start up the server again.

Command Line

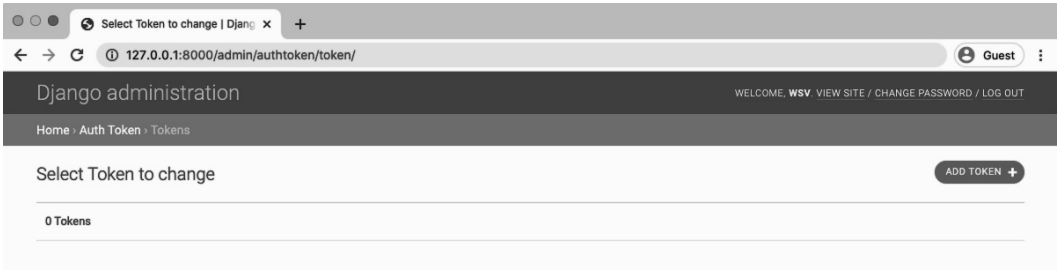
```
(blogapi) $ python manage.py runserver
```

Nếu bạn điều hướng đến quản trị viên Django tại http://127.0.0.1:8000/admin/ bạn sẽ thấy hiện có phần Mã thông báo ở trên cùng. Đảm bảo rằng bạn đã đăng nhập bằng tài khoản siêu người dùng của mình để có quyền truy cập.



Admin Homepage with Tokens

Click on the link for `Tokens`. Currently there are no tokens which might be surprising.



Admin Tokens Page

Sau tất cả, chúng tôi có người dùng hiện tại. Tuy nhiên, các token chỉ được tạo *sau khi* có lệnh gọi API để người dùng đăng nhập. Chúng tôi chưa làm điều đó nên không có mã thông báo. Chúng tôi sẽ sớm thôi!

Endpoints

Chúng tôi cũng cần tạo điểm cuối để người dùng có thể đăng nhập và đăng xuất. Chúng tôi *có thể* tạo một ứng dụng dành riêng cho người dùng cho mục đích này và sau đó thêm url, chế độ xem và bộ nối tiếp hóa của riêng chúng tôi. Tuy nhiên, xác thực người dùng là một lĩnh vực mà chúng tôi thực sự không muốn phạm sai lầm. Và vì hầu hết tất cả các API đều yêu cầu chức năng này, nên có nghĩa là có một số gói của bên thứ ba tuyệt vời và đã được thử nghiệm mà chúng tôi có thể sử dụng nó để thay thế.

Đáng chú ý là chúng tôi sẽ sử dụng dj-rest-auth[83] kết hợp với django-allauth[84] để đơn giản hóa mọi thứ. Đừng cảm thấy tồi tệ khi sử dụng các gói của bên thứ ba. Chúng tồn tại vì một lý do và ngay cả những chuyên gia Django giỏi nhất cũng dựa vào chúng mọi lúc. Không có ích gì khi phát minh lại bánh xe nếu bạn không cần phải làm vậy!

dj-rest-auth

Đầu tiên chúng ta sẽ thêm các điểm cuối API đăng nhập, đăng xuất và đặt lại mật khẩu. Chúng ra khỏi hộp với gói dj-rest-auth phổ biến. Dừng máy chủ với Control+c và sau đó cài đặt nó.

Command Line

```
(blogapi) $ pipenv install dj-rest-auth==1.1.0
```

Add the new app to the `INSTALLED_APPS` config in our `config/settings.py` file.

Code

```
# config/settings.py
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',

    # 3rd-party apps
    'rest_framework',
    'rest_framework.authtoken',
    'dj_rest_auth', # new

    # Local
    'posts',
]
```

[83] https://github.com/jazzband/dj-rest-auth
[84] https://github.com/pennersr/django-allauth

Cập nhật tệp config/urls.py của chúng tôi với gói dj_rest_auth. Chúng ta đang thiết lập các route URL thành api/v1/dj-rest-auth. Đảm bảo lưu ý rằng URL phải có dấu gạch ngang - không phải dấu gạch dưới
_, which is an easy mistake to make.

Code

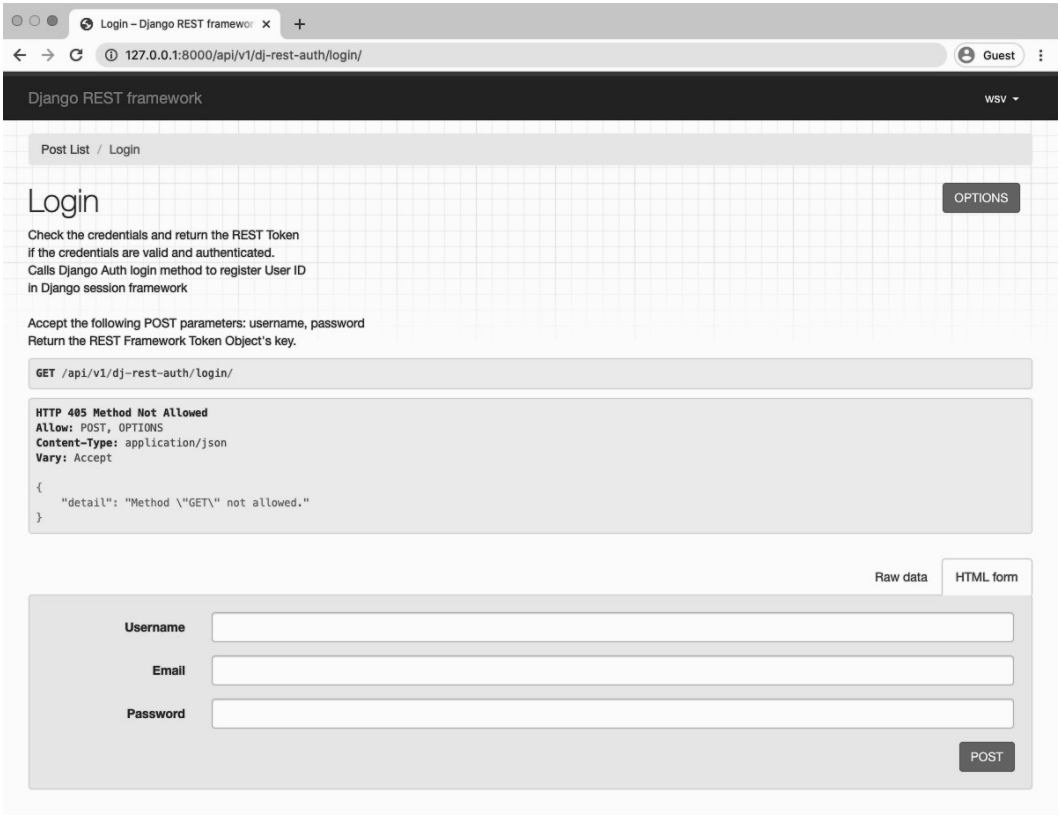
```
# config/urls.py
from django.contrib import admin
from django.urls import include, path

urlpatterns = [
    path('admin/', admin.site.urls),
    path('api/v1/', include('posts.urls')),
    path('api-auth/', include('rest_framework.urls')),
    path('api/v1/dj-rest-auth/', include('dj_rest_auth.urls')), # new
]
```

Và chúng tôi đã hoàn tất! Nếu bạn đã từng cố gắng triển khai các điểm cuối xác thực người dùng của riêng mình, thì thực sự đáng kinh ngạc khi dj-rest-auth tiết kiệm được bao nhiêu thời gian — và đau đầu — dj-rest-auth cho chúng tôi. Bây giờ chúng ta có thể quay lên máy chủ để xem những gì dj-rest-auth đã cung cấp.

Command Line

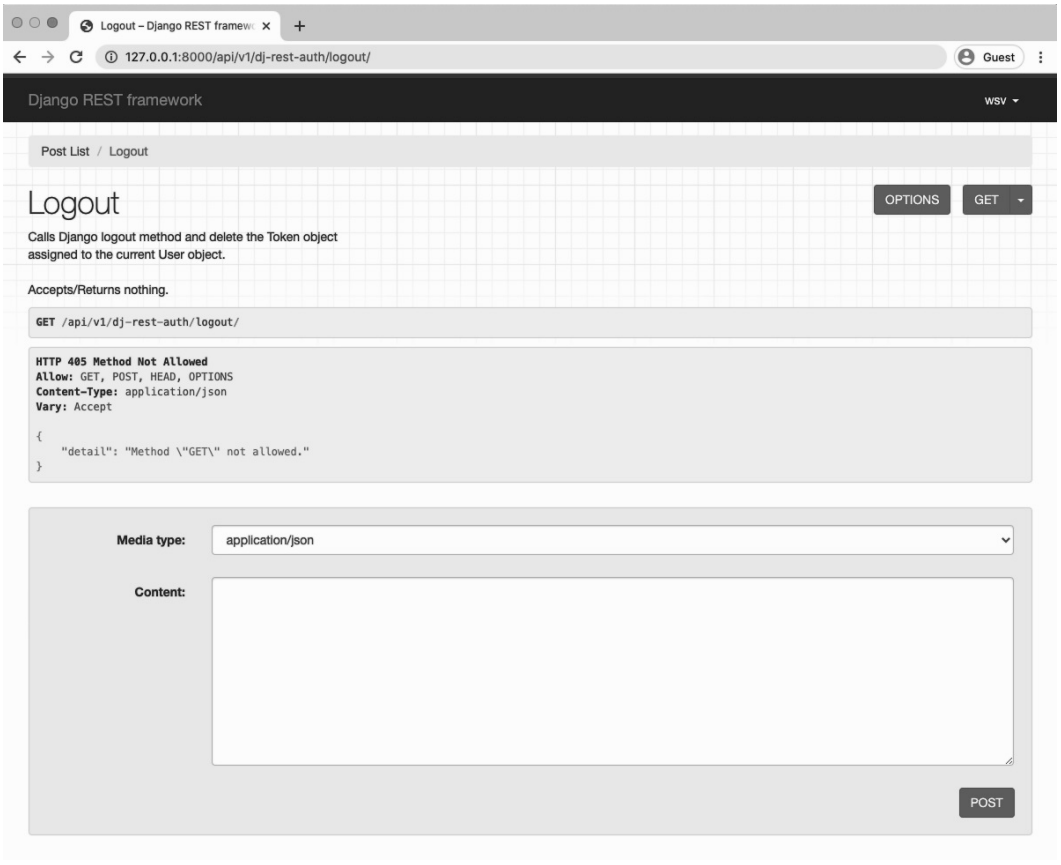
```
(blogapi) $ python manage.py runserver
```

We have a working log in endpoint at `http://127.0.0.1:8000/api/v1/dj-rest-auth/login/`.



API Log In Endpoint

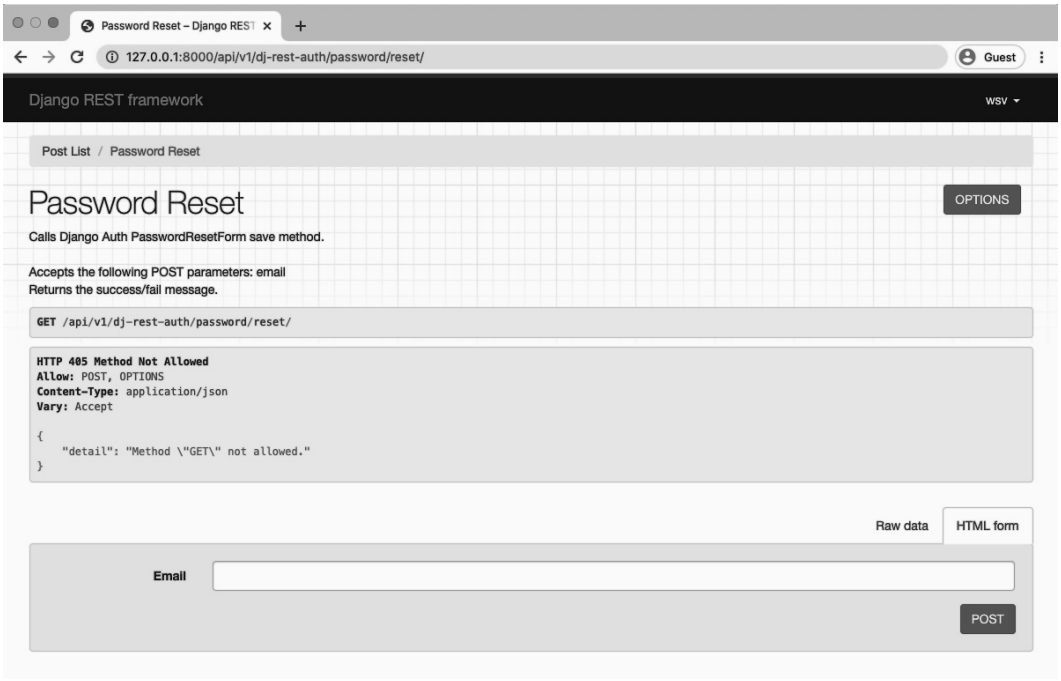
And a log out endpoint at `http://127.0.0.1:8000/api/v1/dj-rest-auth/logout/`.



API Log Out Endpoint

There are also endpoints for password reset, which is located at:

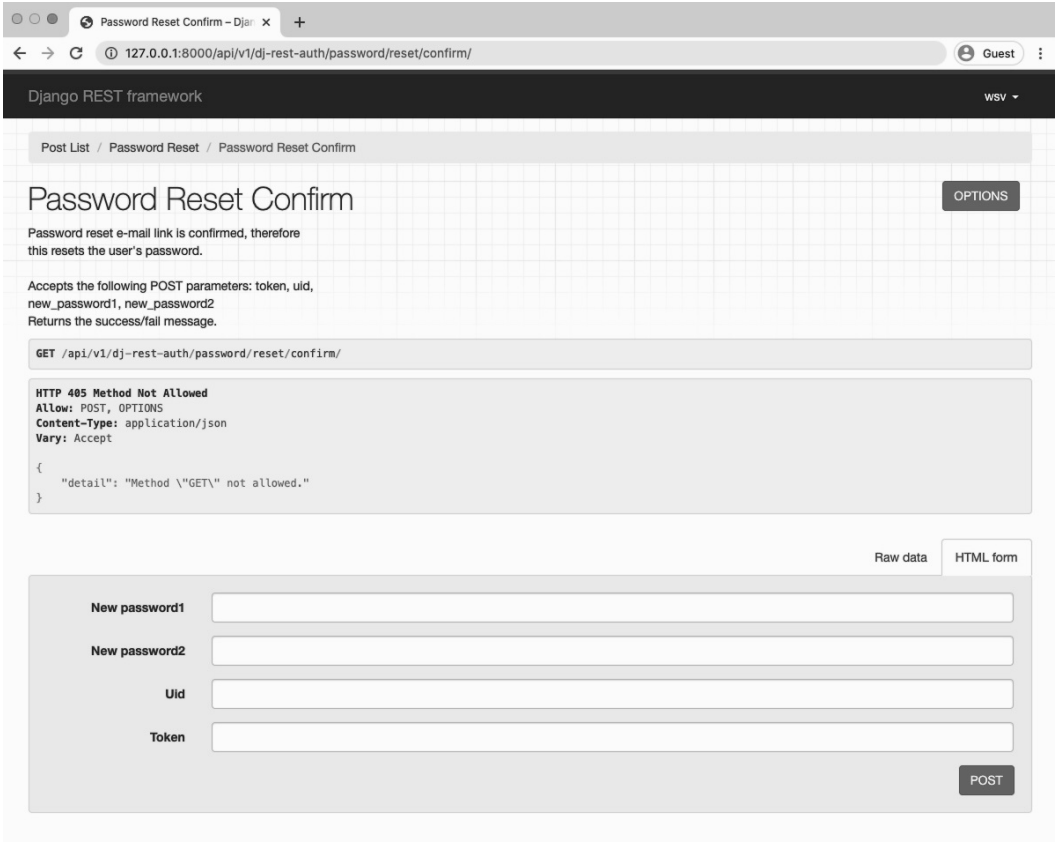
```
http://127.0.0.1:8000/api/v1/dj-rest-auth/password/reset
```



API Password Reset

And for password reset confirmed:

```
http://127.0.0.1:8000/api/v1/dj-rest-auth/password/reset/confirm
```



API Password Reset Confirm

User Registration

Tiếp theo là đăng ký người dùng của chúng tôi, hoặc đăng ký, điểm cuối. Django truyền thống không đi kèm với các chế độ xem hoặc URL tích hợp để đăng ký người dùng và Khung Django REST cũng vậy. Có nghĩa là chúng ta cần phải viết mã của riêng mình từ đầu; một cách tiếp cận hơi rủi ro do mức độ nghiêm trọng - và ý nghĩa bảo mật - của việc làm sai điều này.

Một cách tiếp cận phổ biến là sử dụng gói của bên thứ ba django-allauth[85] đi kèm với đăng ký người dùng cũng như một số tính năng bổ sung cho hệ thống xác thực Django như xã hội

[85] https://github.com/pennersr/django-allauth

xác thực qua Facebook, Google, Twitter, v.v. Nếu chúng ta thêm dj_rest_auth.registration từ gói dj-rest-auth thì chúng ta cũng có các điểm cuối đăng ký người dùng!

Stop the local server with `Control+c` and install `django-allauth`.

Command Line

```
(blogapi) $ pipenv install django-allauth~=0.42.0
```

Sau đó, cập nhật cài đặt INSTALLED_APPS của chúng tôi. Chúng ta phải thêm một số cấu hình mới:

- `django.contrib.sites`
- `allauth`
- `allauth.account`
- `allauth.socialaccount`
- `dj_rest_auth.registration`

Đảm bảo cũng bao gồm EMAIL_BACKEND và SITE_ID. Về mặt kỹ thuật, không quan trọng chúng được đặt ở đâu trong tệp cấu hình / cài đặt.py, nhưng người ta thường thêm các cấu hình bổ sung như vậy ở dưới cùng.

Code

```
# config/settings.py
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'django.contrib.sites', # new

    # 3rd-party apps
    'rest_framework',
    'rest_framework.authtoken',
    'allauth', # new
    'allauth.account', # new
    'allauth.socialaccount', # new
    'dj_rest_auth',
```

```
    'dj_rest_auth.registration', # new

    # Local
    'posts',
]


EMAIL_BACKEND = 'django.core.mail.backends.console.EmailBackend' # new

SITE_ID = 1 # new
```

Cấu hình back-end của email là cần thiết vì theo mặc định, một email sẽ được gửi khi người dùng mới được đăng ký, yêu cầu họ xác nhận tài khoản của mình. Thay vì *cũng* thiết lập một máy chủ email, chúng tôi sẽ xuất email đến bảng điều khiển bằng bảng điều khiển. Cài đặt EmailBackend.

SITE_ID là một phần của khung "trang web" Django [86] tích hợp sẵn, đây là một cách để lưu trữ nhiều trang web từ cùng một dự án Django. Rõ ràng là chúng tôi chỉ có một trang web mà chúng tôi đang làm việc ở đây nhưng django-allauth sử dụng khung trang web, vì vậy chúng tôi phải chỉ định cài đặt mặc định.

Ok. Chúng tôi đã thêm các ứng dụng mới để đã đến lúc cập nhật cơ sở dữ liệu.

Command Line

```
(blogapi) $ python manage.py migrate
```

Sau đó, thêm một tuyến URL mới để đăng ký.

[86] https://docs.djangoproject.com/en/3.1/ref/contrib/sites/

Code

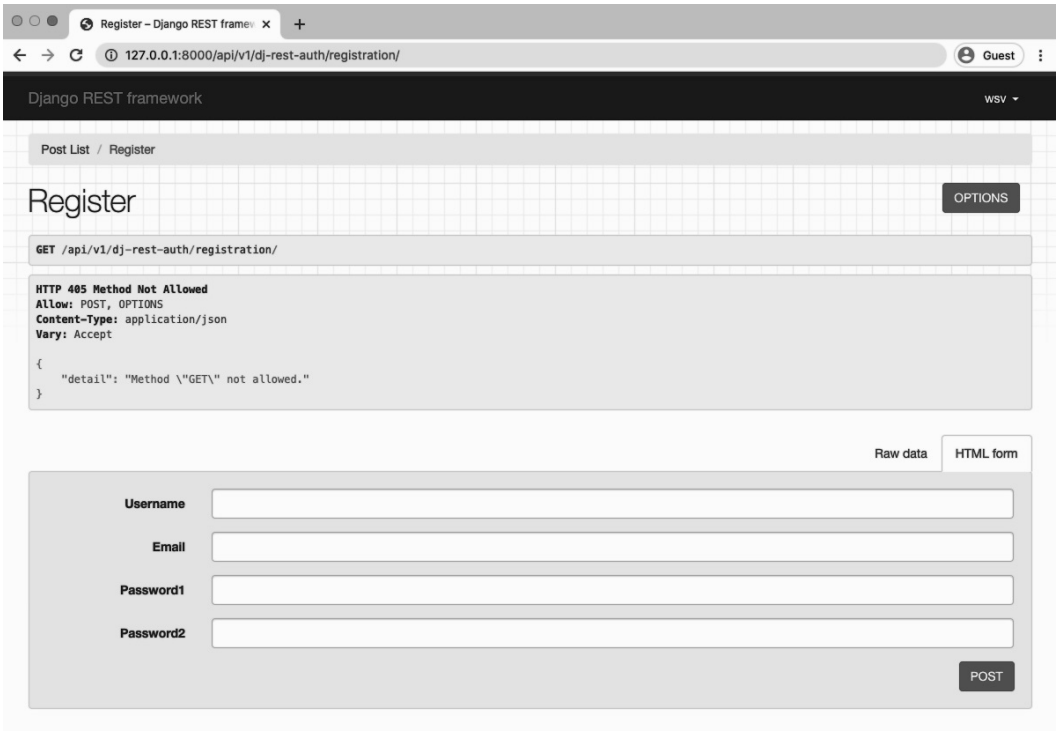
```
# config/urls.py
from django.contrib import admin
from django.urls import include, path

urlpatterns = [
    path('admin/', admin.site.urls),
    path('api/v1/', include('posts.urls')),
    path('api-auth/', include('rest_framework.urls')),
    path('api/v1/dj-rest-auth/', include('dj_rest_auth.urls')),
    path('api/v1/dj-rest-auth/registration/', # new
          include('dj_rest_auth.registration.urls')),
]
```

And we're done. We can run the local server.

Command Line

```
(blogapi) $ python manage.py runserver
```

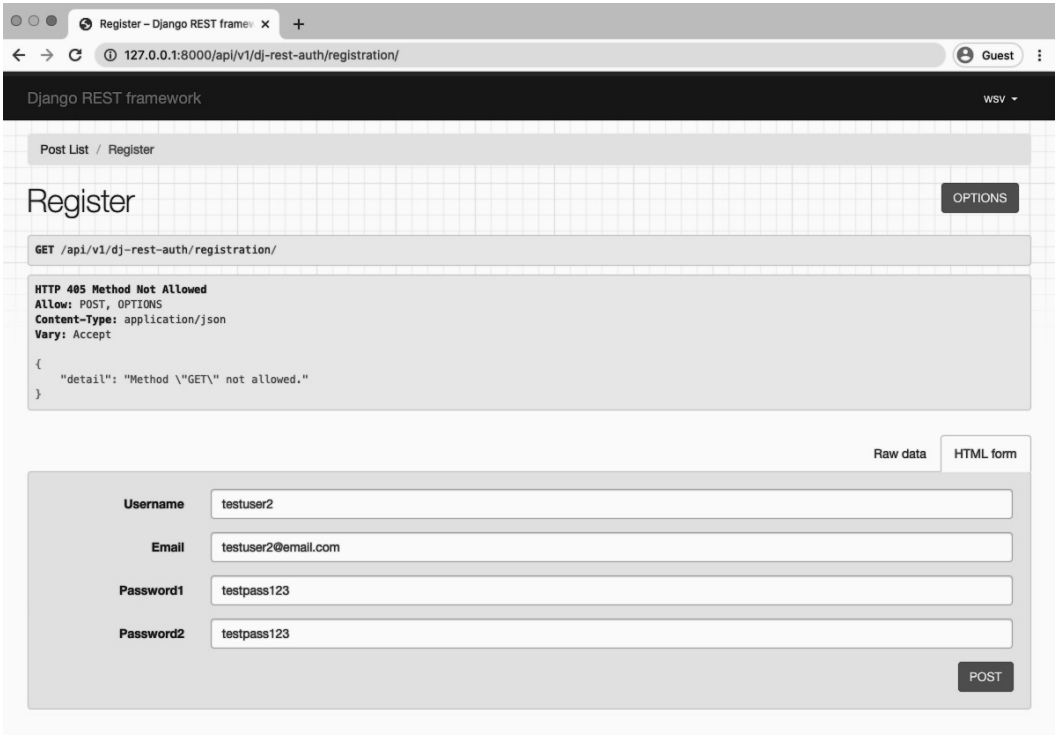
There is now a user registration endpoint at `http://127.0.0.1:8000/api/v1/dj-rest-auth/registration/`.



API Register

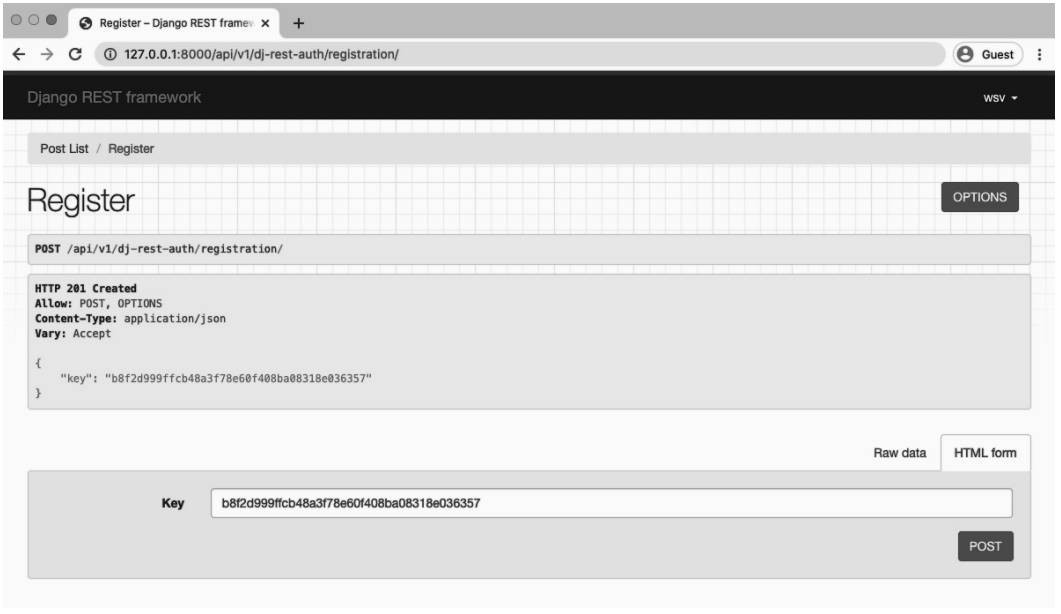
Tokens

Để đảm bảo mọi thứ hoạt động, hãy tạo tài khoản người dùng thứ ba thông qua điểm cuối API có thể duyệt. Tôi đã gọi cho người dùng của mình testuser2. Sau đó nhấp vào nút "ĐĂNG".



API Register New User

Màn hình tiếp theo hiển thị phản hồi HTTP từ máy chủ. Đăng ký người dùng POST của chúng tôi đã thành công, do đó mã trạng thái HTTP 201 Được tạo ở trên cùng. Khóa giá trị trả về làthe auth token cho người dùng mới này.



API Auth Key

Nếu bạn nhìn vào bảng điều khiển dòng lệnh, một email đã được tạo tự động bởi django-allauth. Văn bản mặc định này có thể được cập nhật và một máy chủ email SMTP được thêm vào với cấu hình bổ sung được đề cập trong cuốn sáchDjango for Beginners[87].

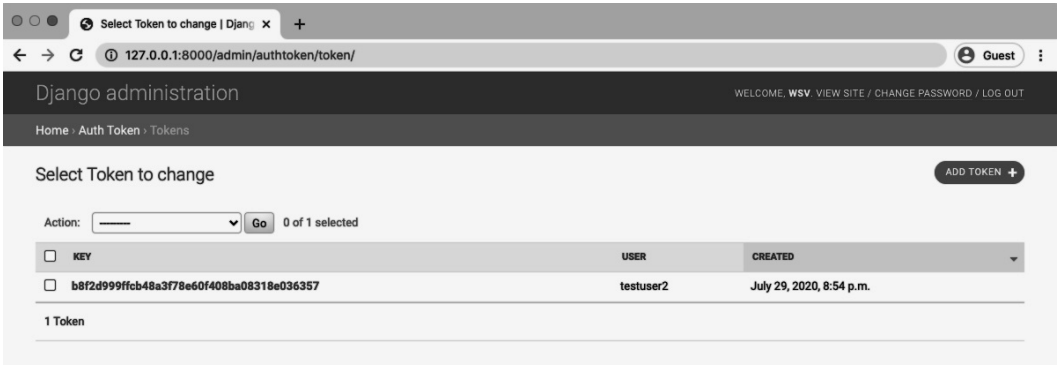
Command Line

```
Content-Type: text/plain; charset="utf-8"
MIME-Version: 1.0
Content-Transfer-Encoding: 7bit
Subject: [example.com] Please Confirm Your E-mail Address
From: webmaster@localhost
To: testuser2@email.com
Date: Wed, 29 Jul 2020 20:54:26 -0000
Message-ID:
 <159605606600.8206.5520712009851546888@1.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0\
.0.0.0.0.0.0.ip6.arpa>

Hello from example.com!

You're receiving this e-mail because user testuser2 has given yours as an e-mail address \
to connect their account.
```

[87] https://djangoforbeginners.com

```
To confirm this is correct, go to http://127.0.0.1:8000/api/v1/dj-rest-auth/\
registration/account-confirm-email/MQ:1k0t5m:6l0l09er1p_cbxgkJWDuSw2j00M/

Thank you from example.com!
example.com
```

Chuyển sang quản trị viên Django trong trình duyệt web của bạn tại http://127.0.0.1:8000/admin/. Bạn sẽ cần sử dụng tài khoản superuser của mình cho việc này. Sau đó nhấp vào liên kết cho Token ở đầu trang. Bạn sẽ được chuyển hướng đến trang Token.

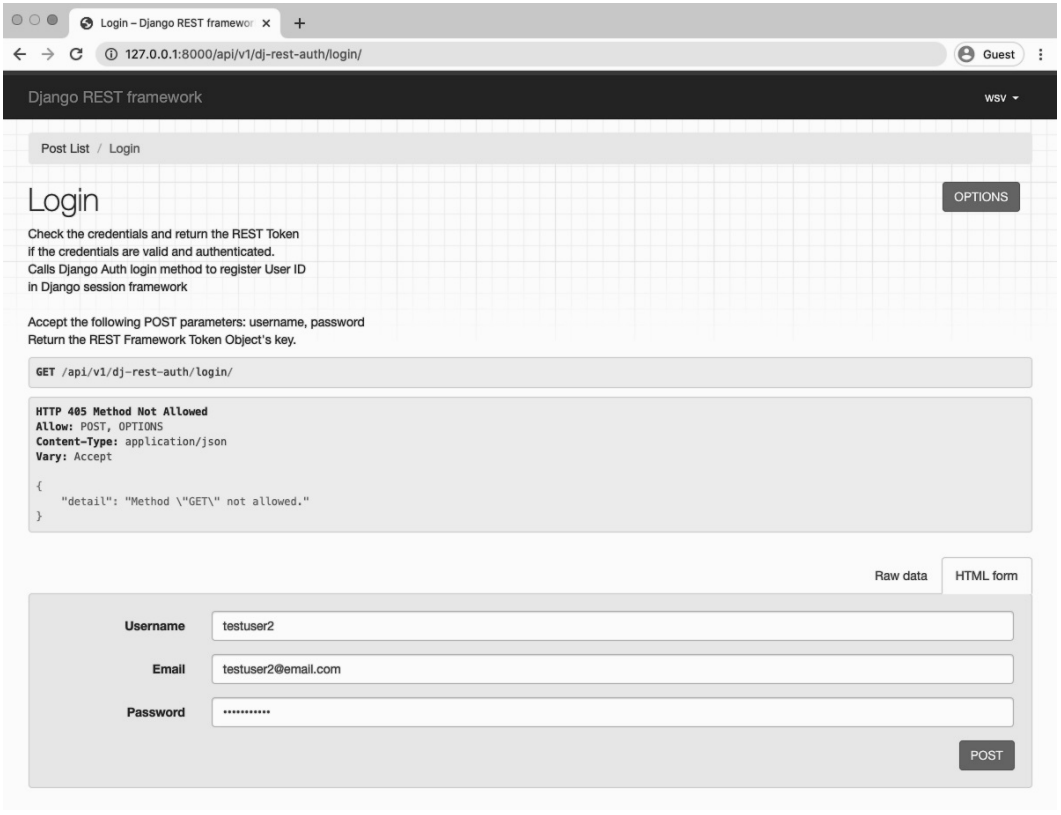


Admin Tokens

Một token duy nhất đã được tạo ra bởi Django REST Framework cho người dùng testuser2. Khi người dùng bổ sung được tạo thông qua API, mã thông báo của họ cũng sẽ xuất hiện ở đây.

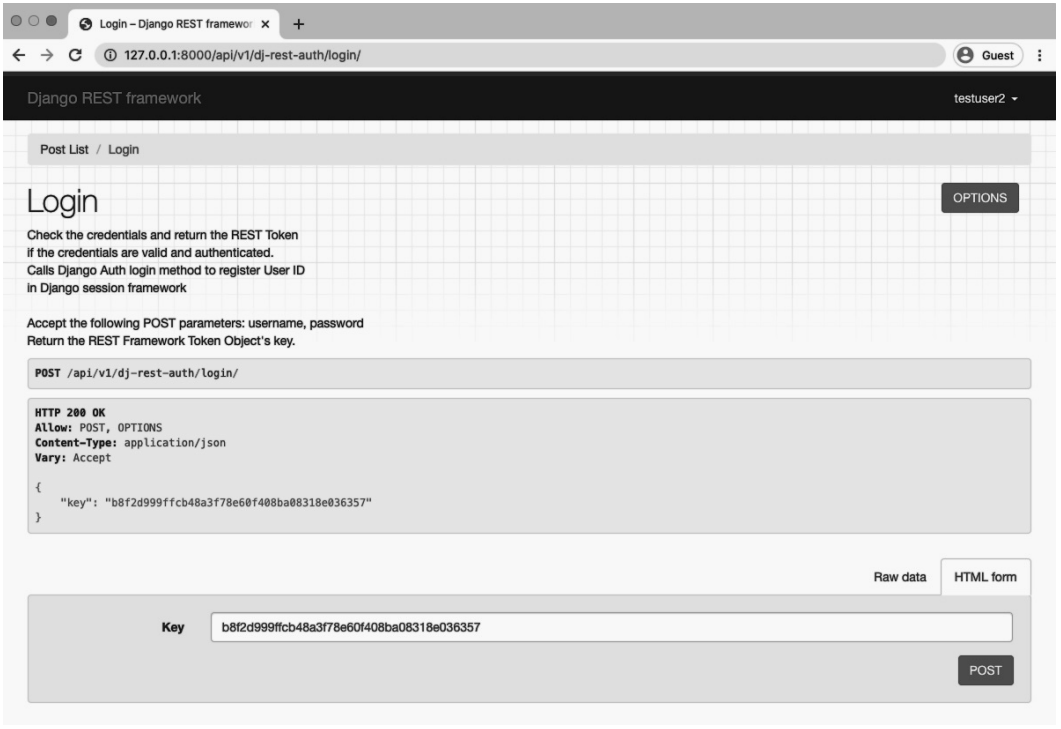
Một câu hỏi hợp lý là, Tại sao không có mã thông báo cho tài khoản superuser hoặc testuser của chúng tôi? Câu trả lời là chúng tôi đã tạo các tài khoản đó trước khi xác thực mã thông báo được thêm vào. Nhưng đừng lo lắng, khi chúng tôi đăng nhập bằng một trong hai tài khoản thông qua API, mã thông báo sẽ tự động được thêm vào và có sẵn.

Tiếp tục, hãy đăng nhập bằng tài khoản testuser2 mới của chúng tôi. Trong trình duyệt web của bạn, hãy điều hướng đến http://127.0.0.1:8000/api/v1/dj-rest-auth/login/. Nhập thông tin cho tài khoản testuser2 của chúng tôi. Nhấp vào nút "ĐĂNG".



API Log In testuser2

Hai điều đã xảy ra. Ở góc trên bên phải, testuser2 tài khoản người dùng của chúng tôi hiển thị, xác nhận rằng chúng tôi hiện đã đăng nhập. Ngoài ra, máy chủ đã gửi lại phản hồi HTTP với mã thông báo.



API Log In Token

Trong framework front-end, chúng ta sẽ cần nắm bắt và lưu trữ token này. Theo truyền thống, điều này xảy ra trên máy khách, trong localStorage[88] hoặc dưới dạng cookie và sau đó tất cả các yêu cầu trong tương lai bao gồm mã thông báo trong tiêu đề như một cách để xác thực người dùng. Lưu ý rằng có thêm các mối quan tâm về bảo mật về chủ đề này, vì vậy bạn nên cẩn thận triển khai các phương pháp hay nhất của khung front-end mà bạn chọn.

Conclusion

Xác thực người dùng là một trong những lĩnh vực khó nắm bắt nhất khi lần đầu tiên làm việc với các API web. Nếu không có lợi ích của cấu trúc nguyên khối, chúng tôi với tư cách là nhà phát triển phải hiểu sâu và định cấu hình các chu kỳ yêu cầu/phản hồi HTTP của mình một cách thích hợp.

Django REST Framework đi kèm với rất nhiều hỗ trợ tích hợp cho quá trình này, bao gồm

[88] https://developer.mozilla.org/en-US/docs/Web/API/Window/localStorage

TokenAuthentication tích hợp sẵn. Tuy nhiên, các nhà phát triển phải định cấu hình các khu vực bổ sung như đăng ký người dùng và url / chế độ xem chuyên dụng. Do đó, một cách tiếp cận phổ biến, mạnh mẽ và an toàn là dựa vào các gói của bên thứ ba dj-rest-auth và django-allauth để giảm thiểu số lượng mã chúng ta phải viết từ đầu.

Chapter 8: Viewsets and Routers

Viewsets[89] and routers[90] là những công cụ trong Django REST Framework có thể tăng tốc độ phát triển API. Chúng là một lớp trừu tượng bổ sung trên đầu trang của lượt xem và URL. Lợi ích chính là một bộ chế độ xem duy nhất có thể thay thế nhiều chế độ xem liên quan. Và một bộ định tuyến có thể tự động tạo URL cho nhà phát triển. Trong các dự án lớn hơn với nhiều điểm cuối, điều này có nghĩa là nhà phát triển phải viết ít mã hơn. Nó cũng, được cho là, dễ dàng hơn cho một nhà phát triển có kinh nghiệm để hiểu và suy luận về một số lượng nhỏ các kết hợp bộ định tuyến và bộ định tuyến hơn là một danh sách dài các chế độ xem và URL riêng lẻ.

Trong chương này, chúng ta sẽ thêm hai điểm cuối API mới vào dự án hiện có của mình và xem việc chuyển đổi từ chế độ xem và URL sang bộ chế độ xem và bộ định tuyến có thể đạt được chức năng tương tự với ít mã hơn nhiều.

User endpoints

Hiện tại chúng ta có các điểm cuối API sau trong dự án của mình. Tất cả chúng đều có tiền tốwith `api/v1/` không được hiển thị cho ngắn gọn:

[89] http://www.django-rest-framework.org/api-guide/viewsets/
[90] http://www.django-rest-framework.org/api-guide/routers/

Diagram

```
|Endpoint                                |HTTP Verb|
|----------------------------------------|---------|
|/                                       |GET      |
|/:pk/                                   |GET      |
|/rest-auth/registration                 |POST     |
|/rest-auth/login                        |POST     |
|/rest-auth/logout                       |GET      |
|/rest-auth/password/reset               |POST     |
|/rest-auth/password/reset/confirm       |POST     |
```

Hai điểm cuối đầu tiên được tạo bởi chúng tôi trong khi dj-rest-auth cung cấp năm điểm cuối khác. Bây giờ chúng ta hãy thêm hai điểm cuối bổ sung để liệt kê tất cả người dùng và người dùng cá nhân. Đây là một tính năng phổ biến trong nhiều API và nó sẽ làm rõ hơn lý do tại sao việc tái cấu trúc lượt xem và URL của chúng tôi thành bộ xem và bộ định tuyến có thể có ý nghĩa.

Django truyền thống có một lớp mô hình Người dùng tích hợp sẵn mà chúng ta đã sử dụng trong chương trước để xác thực. Vì vậy, chúng ta không cần phải tạo một mô hình cơ sở dữ liệu mới. Thay vào đó, chúng ta chỉ cần kết nối các điểm cuối mới. Quá trình này *luôn* bao gồm ba bước sau:

- new serializer class for the model
- new views for each endpoint
- new URL routes for each endpoint

Bắt đầu với serializer của chúng tôi. Chúng tôi cần nhập khẩuthe `User` mô hình và sau đó tạo một`UserSerializer`
cl ass mà sử dụng nó. Sau đó thêm nó vào hiện có của chúng tôi`posts/serializers.py` file.

Code

```
# posts/serializers.py
from django.contrib.auth import get_user_model # new
from rest_framework import serializers
from .models import Post


class PostSerializer(serializers.ModelSerializer):

    class Meta:
        model = Post
        fields = ('id', 'author', 'title', 'body', 'created_at',)


class UserSerializer(serializers.ModelSerializer): # new

    class Meta:
        model = get_user_model()
        fields = ('id', 'username',)
```

Điều đáng chú ý là trong khi chúng ta đã sử dụng get_user_model để tham khảo mô hình User ở đây, thực tế có ba cách khác nhau để tham chiếu[91] mô hình User trong Django.

Bằng cách sử dụng get_user_model chúng tôi đảm bảo rằng chúng tôi đang đề cập đến mô hình người dùng chính xác, cho dù đó là Mô hình người dùng mặc định hay mô hình người dùng tùy chỉnh [92] như thường được xác định trong các dự án Django mới.

Tiếp tục, chúng ta cần xác định các chế độ xem cho *từng* điểm cuối. Đầu tiên thêm UserSerializer vào danh sách nhập. Sau đó, tạo cả lớp UserList liệt kê ra tất cả người dùng và lớp UserDetail cung cấp chế độ xem chi tiết của một người dùng cá nhân. Cũng giống như với chế độ xem bài đăng của chúng tôi, chúng tôi có thể sử dụng ListCreateAPIView và RetrieveUpdateDestroyAPIView tại đây. Chúng tôi cũng cần tham khảo mô hình người dùng qua get_user_model để nó được nhập ở dòng trên cùng.

[91] https://docs.djangoproject.com/en/3.1/topics/auth/customizing/#referencing-the-user-model
[92] https://docs.djangoproject.com/en/3.1/topics/auth/customizing/#specifying-a-custom-user-model

Code

```
# posts/views.py
from django.contrib.auth import get_user_model # new
from rest_framework import generics
from .models import Post
from .permissions import IsAuthorOrReadOnly
from .serializers import PostSerializer, UserSerializer # new


class PostList(generics.ListCreateAPIView):
    queryset = Post.objects.all()
    serializer_class = PostSerializer


class PostDetail(generics.RetrieveUpdateDestroyAPIView):
    permission_classes = (IsAuthorOrReadOnly,)
    queryset = Post.objects.all()
    serializer_class = PostSerializer


class UserList(generics.ListCreateAPIView): # new
    queryset = get_user_model().objects.all()
    serializer_class = UserSerializer


class UserDetail(generics.RetrieveUpdateDestroyAPIView): # new
    queryset =  get_user_model().objects.all()
    serializer_class = UserSerializer
```

Nếu bạn nhận thấy, có khá nhiều sự lặp lại ở đây. Cả Chế độ xem bài đăng và Chế độ xem người dùng đều có cùng một tập truy vấn và serializer_class. Có lẽ chúng có thể được kết hợp theo một cách nào đó để tiết kiệm mã?

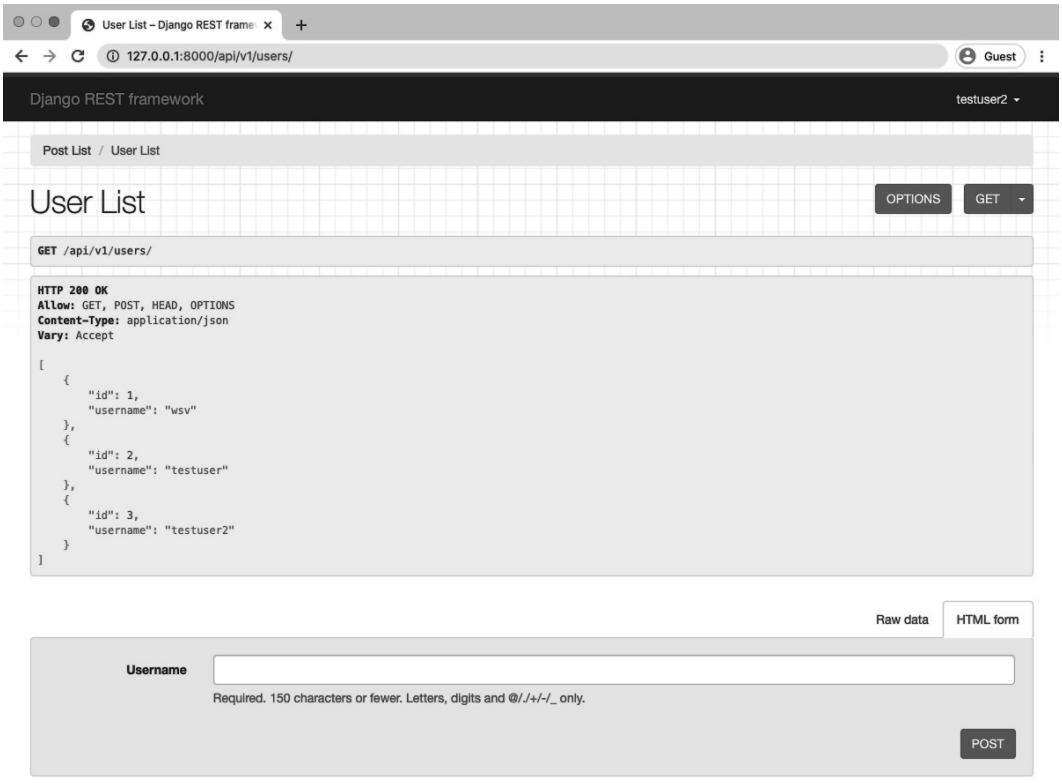
Cuối cùng, chúng tôi có các tuyến URL của chúng tôi. Đảm bảo nhập các chế độ xem UserList và UserDetail mới của chúng tôi. Sau đó, chúng ta có thể sử dụng tiền tố users/ cho mỗi.

Code

```
# posts/urls.py
from django.urls import path
from .views import UserList, UserDetail, PostList, PostDetail # new

urlpatterns = [
    path('users/', UserList.as_view()), # new
    path('users/<int:pk>/', UserDetail.as_view()), # new
    path('', PostList.as_view()),
    path('<int:pk>/', PostDetail.as_view()),
]
```

And we're done. Make sure the local server is still running and jump over to the browsable API to confirm everything works as expected.

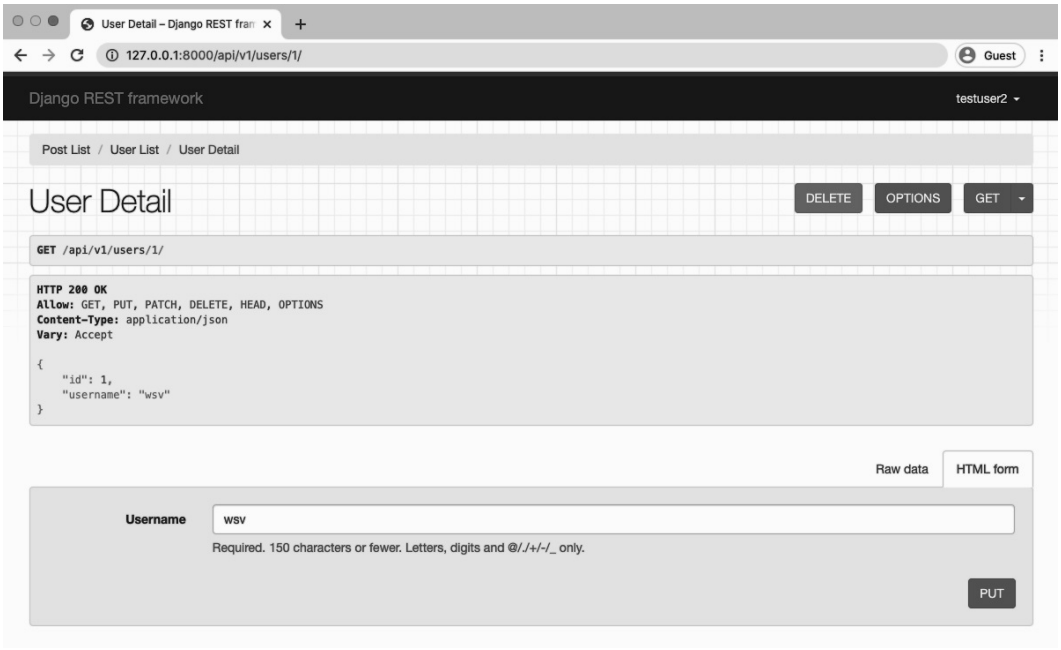
Our user list endpoint is located at `http://127.0.0.1:8000/api/v1/users/`.



API Users List

The status code is `200 OK` which means everything is Làm việc. We can see our three existing users.

Điểm cuối chi tiết người dùng có sẵn tại khóa chính cho mỗi người dùng. Vì vậy, tài khoản superuser của chúng tôi được đặt tại: http://127.0.0.1:8000/api/v1/users/1/.



API User Instance

Viewsets

Một tập hợp chế độ xem là một cách để kết hợp logic cho nhiều chế độ xem liên quan vào một lớp duy nhất. Nói cách khác, một bộ chế độ xem có thể thay thế nhiều chế độ xem. Hiện tại, chúng tôi có bốn chế độ xem: **hai cho các bài đăng trên blog và hai cho người dùng.** Thay vào đó, chúng tôi có thể bắt chước cùng một chức năng với hai bộ chế độ xem: một cho các bài đăng trên blog và một cho người dùng.

Sự đánh đổi là có sự mất khả năng đọc đối với các nhà phát triển đồng nghiệp, những người *không* quen thuộc với các chế độ xem. Vì vậy, đó là một sự đánh đổi.

Đây là giao diện của code trong tập tin post/views.py cập nhật của chúng ta khi chúng ta hoán đổi trong viewsets.

Code

```
# posts/views.py
from django.contrib.auth import get_user_model
from rest_framework import viewsets # new
from .models import Post
from .permissions import IsAuthorOrReadOnly
from .serializers import PostSerializer, UserSerializer


class PostViewSet(viewsets.ModelViewSet): # new
    permission_classes = (IsAuthorOrReadOnly,)
    queryset = Post.objects.all()
    serializer_class = PostSerializer


class UserViewSet(viewsets.ModelViewSet): # new
    queryset = get_user_model().objects.all()
    serializer_class = UserSerializer
```

Ở trên cùng thay vì nhập thuốc generic từ rest_framework chúng ta hiện đang nhập các viewset trên dòng thứ hai. Sau đó, chúng tôi đang sử dụng ModelViewSet[93] cung cấp cả chế độ xem danh sách và chế độ xem chi tiết cho chúng tôi. Và chúng tôi không còn phải lặp lại cùng một tập truy vấn và serializer_class cho mỗi chế độ xem như chúng tôi đã làm trước đây!

Tại thời điểm này, máy chủ web cục bộ sẽ dừng lại khi Django phàn nàn về việc thiếu các đường dẫn URL tương ứng. Hãy thiết lập những điều đó tiếp theo.

Routers

Bộ định tuyến[94] hoạt động trực tiếp với các bộ chế độ xem để tự động tạo mẫu URL cho chúng tôi. Tệp bài đăng / url.py hiện tại của chúng tôi có bốn mẫu URL: hai mẫu cho bài đăng trên blog và hai mẫu cho người dùng. Thay vào đó, chúng ta có thể áp dụng một tuyến đường duy nhất cho mỗi viewset. Vì vậy, hai tuyến đường thay vì bốn mẫu URL. Điều đó nghe có vẻ tốt hơn, phải không?

[93] http://www.django-rest-framework.org/api-guide/viewsets/#modelviewset
[94] http://www.django-rest-framework.org/api-guide/routers/

Django REST Framework có hai bộ định tuyến mặc định: SimpleRouter[95] và DefaultRouter[96]. Chúng tôi sẽ sử dụng SimpleRouter nhưng cũng có thể tạo các bộ định tuyến tùy chỉnh cho chức năng nâng cao hơn.

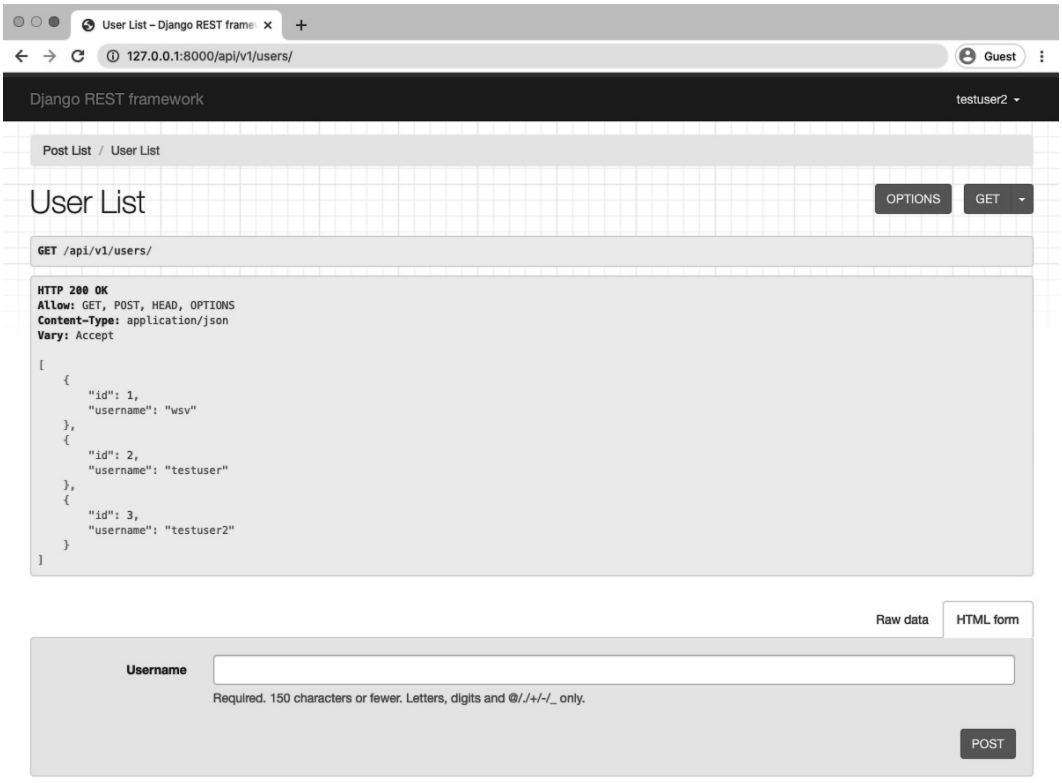
Đây là giao diện của code được cập nhật:

Code

```
# posts/urls.py
from django.urls import path
from rest_framework.routers import SimpleRouter
from .views import UserViewSet, PostViewSet

router = SimpleRouter()
router.register('users', UserViewSet, basename='users')
router.register('', PostViewSet, basename='posts')

urlpatterns = router.urls
```

Trên dòng trên cùng SimpleRouter được nhập, cùng với quan điểm của chúng tôi. Bộ định tuyến được đặt thành SimpleRouter và chúng tôi "đăng ký" từng viewset cho Người dùng và Bài đăng. Cuối cùng, chúng tôi đặt URL của mình để sử dụng bộ định tuyến mới. Hãy tiếp tục và kiểm tra bốn endpoint của chúng ta ngay bây giờ bằng cách khởi động local server với python manage.py runserver.

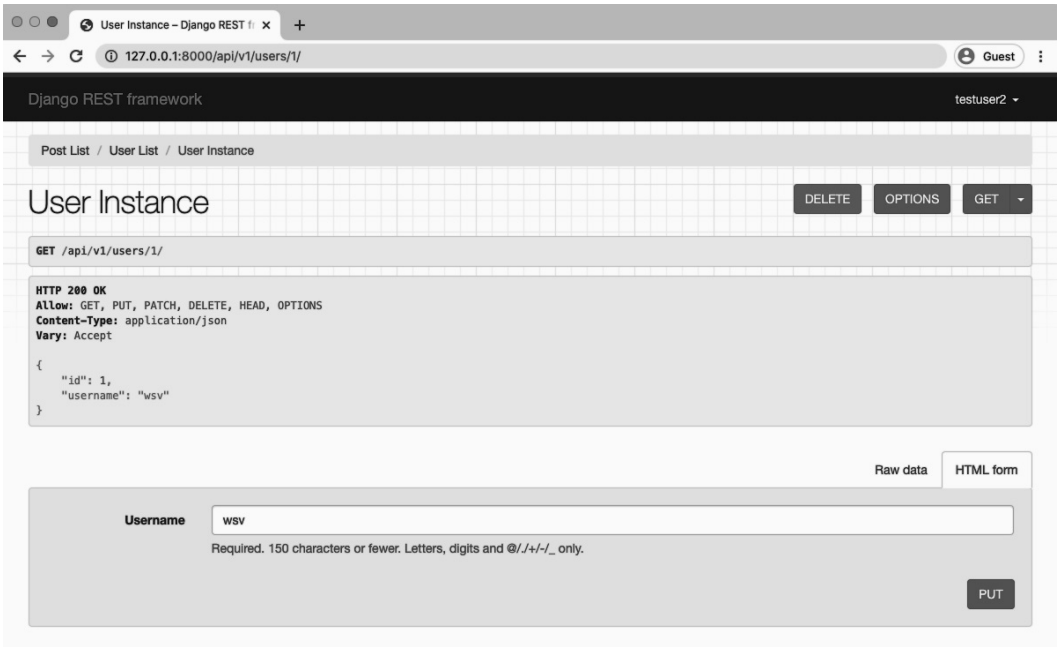
[95] http://www.django-rest-framework.org/api-guide/routers/#simplerouter
[96] http://www.django-rest-framework.org/api-guide/routers/#defaultrouter



API User List

Lưu ý rằng Danh sách người dùng giống nhau, tuy nhiên chế độ xem chi tiết hơi khác một chút. Bây giờ nó được gọi là“User Instance” instead of “User Detail” và có một tùy chọn "xóa" bổ sung được tích hợp sẵn trong ModelViewSet[97].

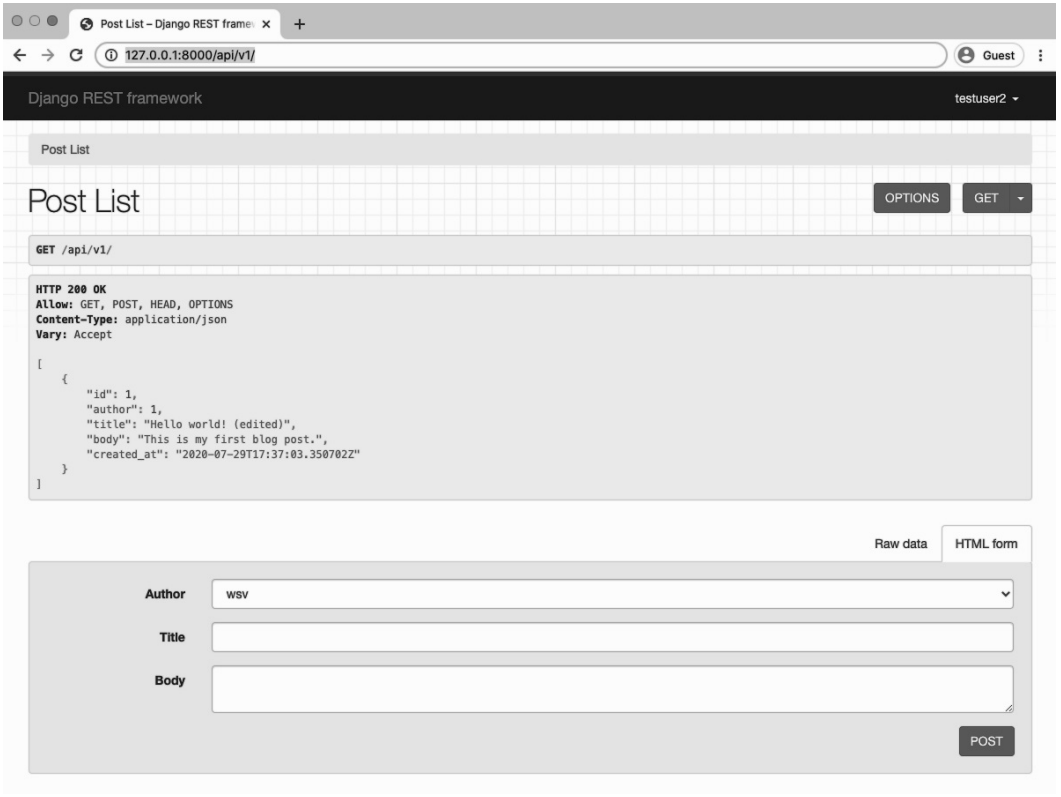
[97] http://www.django-rest-framework.org/api-guide/viewsets/#modelviewset



API User Detail

Có thể tùy chỉnh các chế độ xem nhưng một sự đánh đổi quan trọng để đổi lấy việc viết mã ít hơn một chút với các bộ chế độ xem là cài đặt mặc định có thể yêu cầu một số cấu hình bổ sung để phù hợp chính xác với những gì bạn muốn.

Moving along to the `Post List` at `http://127.0.0.1:8000/api/v1/` we can see it is the same:

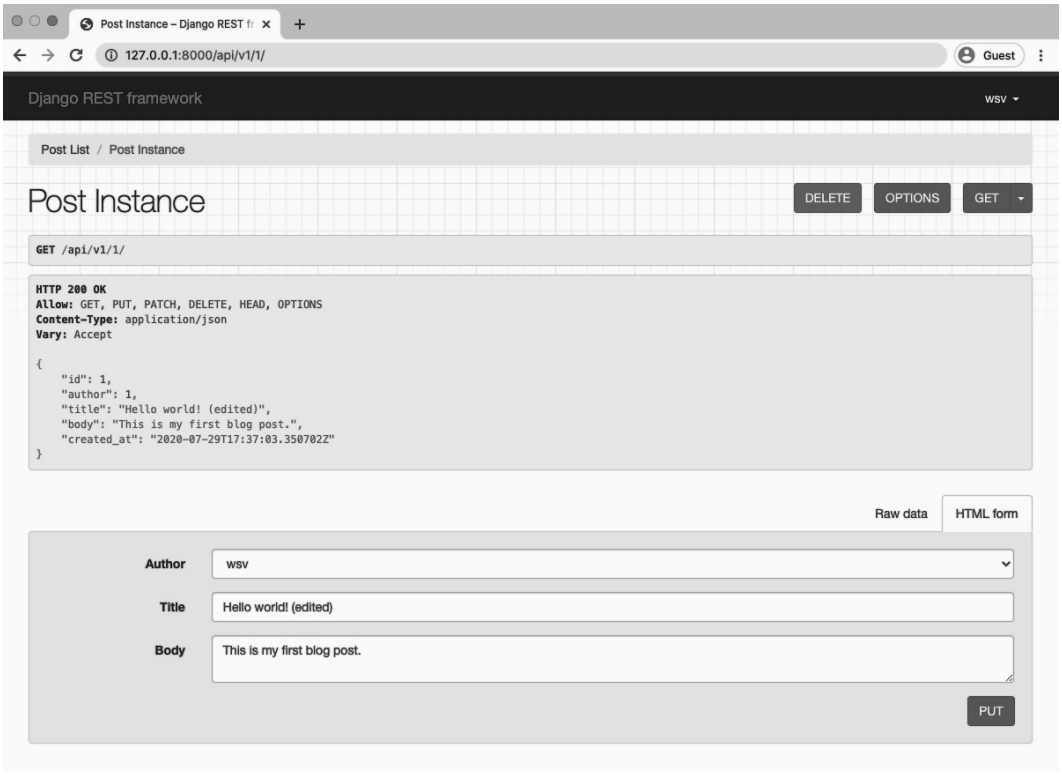


API Post List

Và quan trọng là các quyền của chúng tôi vẫn hoạt động. Khi đăng nhập bằng tài khoản testuser2 của chúng ta, Post Instance tại http://127.0.0.1:8000/api/v1/1/ là dạng chỉ đọc.

API Post Instance Not Owner

Tuy nhiên, nếu chúng tôi đăng nhập bằng tài khoản siêu người dùng của mình, là tác giả của bài đăng trên blog đơn lẻ, thì chúng tôi có đầy đủ đặc quyền đọc-ghi-chỉnh sửa-xóa.



API Post Instance Owner

Conclusion

Bộ xem và bộ định tuyến là một sự trừu tượng mạnh mẽ làm giảm lượng mã mà chúng ta với tư cách là nhà phát triển phải viết. Tuy nhiên, sự ngắn gọn này phải trả giá bằng một đường cong học tập ban đầu. Sẽ cảm thấy lạ trong vài lần đầu tiên bạn sử dụng bộ chế độ xem và bộ định tuyến thay vì chế độ xem và mẫu URL.

Cuối cùng, quyết định *khi nào* nên thêm bộ xem và bộ định tuyến vào dự án của bạn là khá chủ quan. Một nguyên tắc chung là bắt đầu với lượt xem và URL. Khi API của bạn phát triển phức tạp nếu bạn thấy mình lặp đi lặp lại các mẫu điểm cuối giống nhau, thì hãy tìm đến bộ chế độ xem và bộ định tuyến. Cho đến lúc đó, hãy giữ mọi thứ đơn giản.

Chapter 9: Schemas and Documentation

Bây giờ chúng ta đã hoàn thành API của mình, chúng ta cần một cách để ghi lại chức năng của nó một cách nhanh chóng và chính xác cho những người khác. Rốt cuộc, trong hầu hết các công ty và nhóm, nhà phát triển đang sử dụng API không phải là cùng một nhà phát triển đã xây dựng nó. May mắn thay, có những công cụ tự động để xử lý việc này cho chúng tôi.

A schema[98] là một tài liệu có thể đọc được bằng máy phác thảo tất cả các điểm cuối API, URL và động từ HTTP có sẵn (`GET`, `POST`, `PUT`, `DELETE`, etc.) họ hỗ trợ. Tài liệu là thứ được thêm vào lược đồ giúp con người đọc và tiêu thụ dễ dàng hơn. Trong chương này, chúng ta sẽ thêm một lược đồ vào dự án *Blog* của chúng ta và sau đó thêm hai cách tiếp cận tài liệu khác nhau. Cuối cùng, chúng tôi sẽ triển khai một cách tự động để ghi lại bất kỳ thay đổi hiện tại và tương lai nào đối với API của chúng tôi.

Xin nhắc lại, đây là danh sách đầy đủ các điểm cuối API hiện tại của chúng tôi:

Diagram

```
|Endpoint                              |HTTP Verb|
|--------------------------------------|---------|
|/                                     |GET      |
|/:pk/                                 |GET      |
|users/                                |GET      |
|users/:pk/                            |GET      |
|/rest-auth/registration               |POST     |
|/rest-auth/login                      |POST     |
|/rest-auth/logout                     |GET      |
|/rest-auth/password/reset             |POST     |
|/rest-auth/password/reset/confirm     |POST     |
```

[98] https://www.django-rest-framework.org/api-guide/schemas/#schema

Schemas

Trước phiên bản 3.9[99], Django REST Framework dựa vào Core API 100 cho các lược đồ nhưng hiện tại nó đã chuyển đổi chắc chắn sang lược đồ OpenAPI[101] (trước đây gọi là Swagger).

Bước đầu tiên là cài đặt cả PyYAML 102 và uritemplate[103]. PyYAML sẽ chuyển đổi lược đồ của chúng ta thành định dạng YAML-basd OpenAPI, trong khi uritemplate thêm tham số vào đường dẫn URL.

Command Line

```
(blogapi) $ pipenv install pyyaml==5.3.1 uritemplate==3.0.1
```

Tiếp theo, chúng tôi được trình bày với một lựa chọn: tạo lược đồ tĩnh hoặc lược đồ động. Nếu API của bạn không thay đổi thường xuyên, lược đồ tĩnh có thể được tạo định kỳ và được phân phát từ các tệp tĩnh để có hiệu suất cao. Tuy nhiên, nếu API của bạn thay đổi khá thường xuyên, bạn có thể xem xét tùy chọn động. Chúng tôi sẽ thực hiện cả hai ở đây.

Đầu tiên, là cách tiếp cận lược đồ tĩnh sử dụng lệnh quản lý generateschema. Chúng ta có thể xuất kết quả ra một tệp có tên openapi-schema.yml.

Command Line

```
(blogapi) $ python manage.py generateschema > openapi-schema.yml
```

Nếu bạn mở tệp đó, nó khá dài và không thân thiện với con người. Nhưng đối với một máy tính, nó được định dạng hoàn hảo.

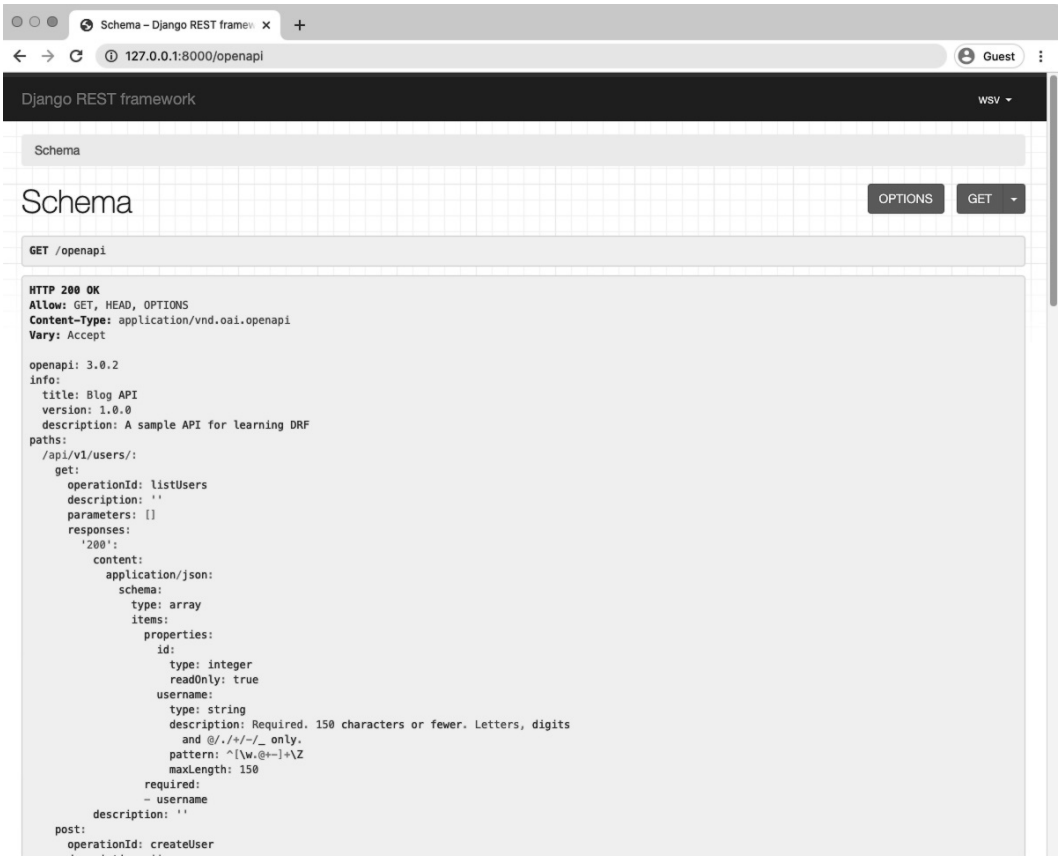
Đối với phương pháp tiếp cận động, cập nhật config/urls.py bằng cách nhập get_schema_view ở trên cùng và sau đó tạo một đường dẫn chuyên dụng tại openapi. Tiêu đề, mô tả và phiên bản có thể được tùy chỉnh khi cần thiết.

[99] https://www.django-rest-framework.org/community/3.9-announcement
[100] http://www.coreapi.org/
[101] https://www.openapis.org/
[102] https://pyyaml.org/
[103] https://github.com/python-hyper/uritemplate

Code

```
# config/urls.py
from django.contrib import admin
from django.urls import include, path
from rest_framework.schemas import get_schema_view # new


urlpatterns = [
    path('admin/', admin.site.urls),
    path('api/v1/', include('posts.urls')),
    path('api-auth/', include('rest_framework.urls')),
    path('api/v1/dj-rest-auth/', include('dj_rest_auth.urls')),
    path('api/v1/dj-rest-auth/registration/',
          include('dj_rest_auth.registration.urls')),
    path('openapi', get_schema_view( # new
        title="Blog API",
        description="A sample API for learning DRF",
        version="1.0.0"
    ), name='openapi-schema'),
]
```

Nếu bạn khởi động lại máy chủ cục bộ bằng python manage.py runserver và điều hướng đến điểm cuối URL lược đồ mới của chúng tôi tại http://127.0.0.1:8000/openapi lược đồ được tạo tự động của toàn bộ API của chúng tôi có sẵn.



API Schema

Cá nhân tôi thích cách tiếp cận năng động trong các dự án.

Documentation

Django REST Framework cũng đi kèm với tính năng tài liệu API[104] tích hợp giúp chuyển lược đồ sang định dạng thân thiện hơn *nhiều* cho các nhà phát triển khác.

Hiện tại, có ba cách tiếp cận phổ biến ở đây: sử dụng SwaggerUI 105, ReDoc[106] hoặc thứ ba-

[104] https://www.django-rest-framework.org/topics/documenting-your-api/

[105] https://swagger.io/tools/swagger-ui/

[106] https://github.com/Rebilly/ReDoc

Gói bên DRF-YASG[107]. Vì drf-yasg khá phổ biến và đi kèm với rất nhiều tính năng tích hợp, chúng tôi sẽ sử dụng nó ở đây.

Bước một là cài đặt phiên bản mới nhất của drf-yasg.

Command Line

```
(blogapi) $ pipenv install drf-yasg==1.17.1
```

Step two, add it to our `INSTALLED_APPS` configuration in `config/settings.py`.

Code

```
# config/settings.py
INSTALLED_APPS = [
    ...
    # 3rd-party apps
    'rest_framework',
    'rest_framework.authtoken',
    'allauth',
    'allauth.account',
    'allauth.socialaccount',
    'rest_auth',
    'rest_auth.registration',
    'drf_yasg', # new

    # Local
    'posts.apps.PostsConfig',
]
```

Bước ba, cập nhật tệp urls.py cấp dự án của chúng tôi. Ở đầu tệp, chúng ta có thể thay thế get_schema_view của DRF bằng drf_yasg từ cũng như nhập openapi. Chúng tôi cũng sẽ thêm quyền của DRF cho các tùy chọn bổ sung.

The `schema_view` Biến được cập nhật và bao gồm các trường bổ sung như `terms_of_service`, `contact`, and `license`. Then under our `urlpatterns` we add paths for both Swagger and ReDoc.

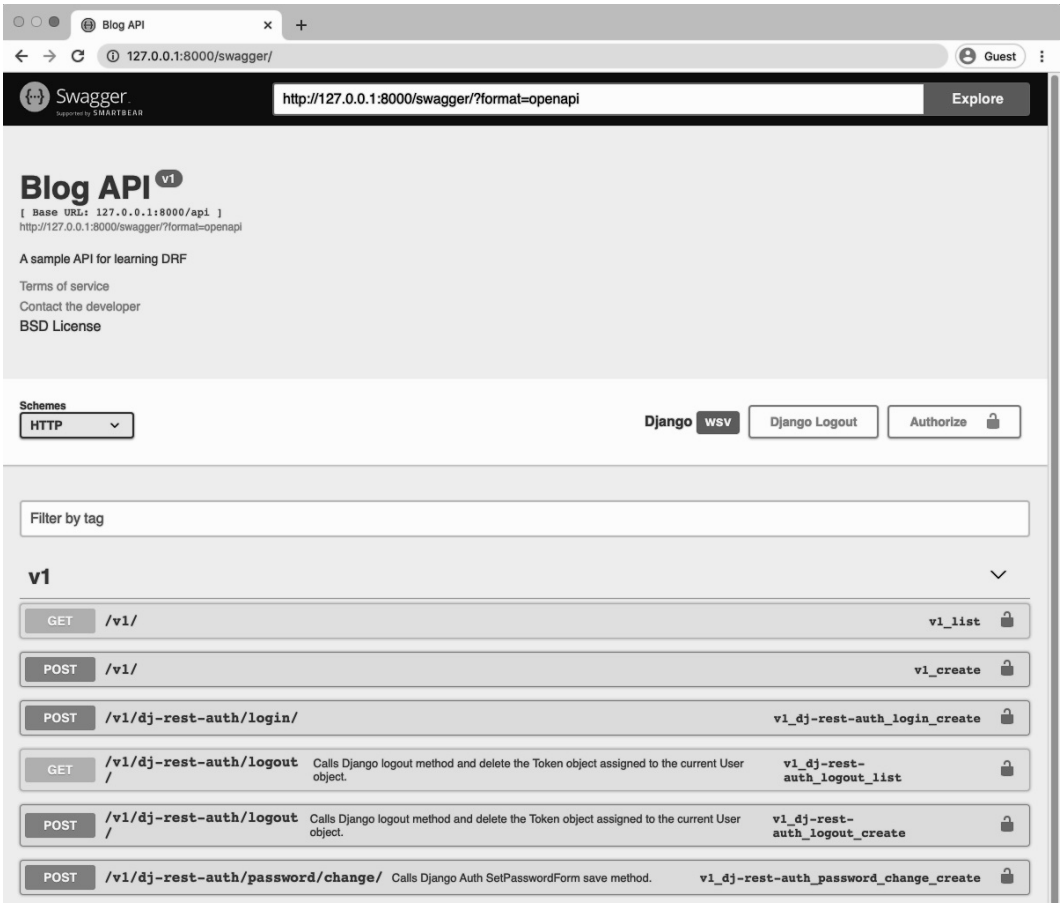
[107] https://drf-yasg.readthedocs.io/en/stable/

Code

```
# config/urls.py
from django.contrib import admin
from django.urls import include, path
from rest_framework import permissions # new
from drf_yasg.views import get_schema_view # new
from drf_yasg import openapi # new

schema_view = get_schema_view( # new
   openapi.Info(
      title="Blog API",
      default_version="v1",
      description="A sample API for learning DRF",
      terms_of_service="https://www.google.com/policies/terms/",
      contact=openapi.Contact(email="hello@example.com"),
      license=openapi.License(name="BSD License"),
   ),
   public=True,
   permission_classes=(permissions.AllowAny,),
)

urlpatterns = [
    path('admin/', admin.site.urls),
    path('api/v1/', include('posts.urls')),
    path('api-auth/', include('rest_framework.urls')),
    path('api/v1/dj-rest-auth/', include('dj_rest_auth.urls')),
    path('api/v1/dj-rest-auth/registration/',
          include('dj_rest_auth.registration.urls')),
    path('swagger/', schema_view.with_ui( # new
      'swagger', cache_timeout=0), name='schema-swagger-ui'),
    path('redoc/', schema_view.with_ui( # new
      'redoc', cache_timeout=0), name='schema-redoc'),
]
```

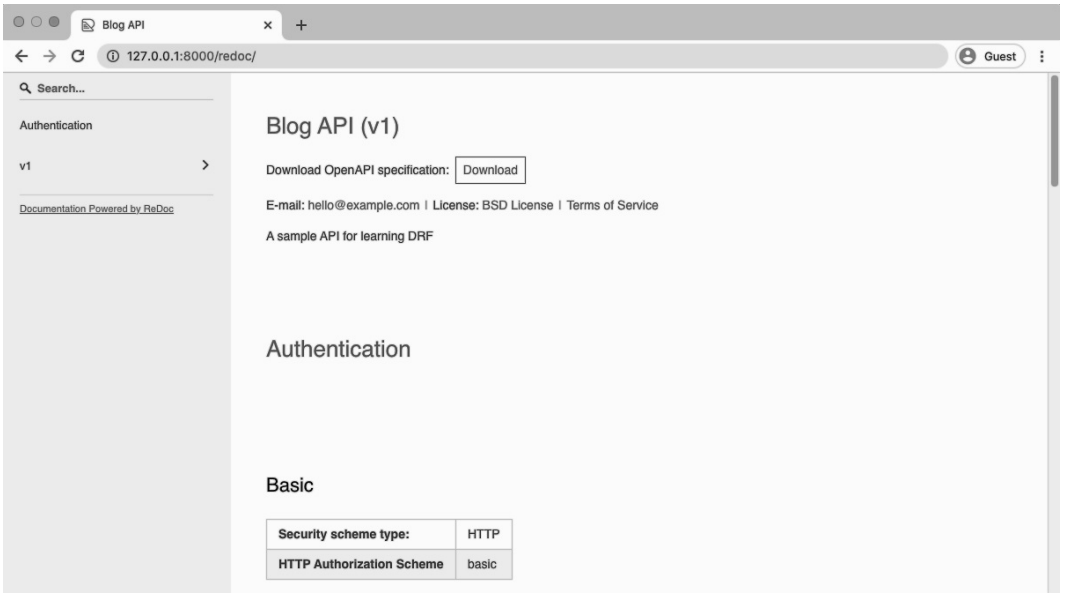
Đảm bảo rằng máy chủ cục bộ đang chạy. Điểm cuối Swagger hiện có sẵn tại:

```
http://127.0.0.1:8000/swagger/
```



Swagger view

Sau đó xác nhận rằng chế độ xem ReDoc cũng đang hoạt động ở http://127.0.0.1:8000/redoc/.



ReDoc view

Tài liệu drf-yasg [108] khá toàn diện và đánh vần nhiều tùy chỉnh khác có thể được thực hiện, tùy thuộc vào nhu cầu của API của bạn.

Conclusion

Thêm tài liệu là một phần quan trọng của bất kỳ API nào. Nó thường là điều đầu tiên mà một nhà phát triển đồng nghiệp xem xét, trong một nhóm hoặc trên một dự án mã nguồn mở. Nhờ các công cụ tự động được đề cập trong chương này, việc đảm bảo API của bạn có tài liệu cập nhật, chính xác chỉ yêu cầu một lượng nhỏ cấu hình.

[108] https://drf-yasg.readthedocs.io/en/stable/readme.html

Conclusion

Bây giờ chúng ta đang ở cuối cuốn sách nhưng chỉ là khởi đầu của những gì có thể hoàn thành với Django REST Framework. Trong suốt ba dự án khác nhau — API *Thư viện*, API *Todo* và API *Blog* — chúng tôi đã xây dựng dần dần các API web phức tạp hơn từ đầu. Và không phải ngẫu nhiên mà ở mỗi bước trên đường đi, Django REST Framework cung cấp các tính năng tích hợp để giúp cuộc sống của chúng ta dễ dàng hơn.

Nếu bạn chưa bao giờ xây dựng API web trước đây với một khuôn khổ khác, hãy cảnh báo rằng bạn đã bị hư hỏng. Và nếu bạn có, hãy yên tâm cuốn sách này chỉ làm trầy xước bề mặt của những gì Django REST Framework có thể làm. Tài liệu chính thức[109] là một nguồn tài nguyên tuyệt vời để khám phá thêm khi bạn đã nắm bắt được những điều cơ bản.

Next Steps

Lĩnh vực lớn nhất đáng để khám phá thêm là thử nghiệm. Các bài kiểm tra Django truyền thống có thể và nên được áp dụng cho bất kỳ dự án API web nào, nhưng cũng có cả một bộ[110] công cụ trong Django REST Framework chỉ để kiểm tra các yêu cầu API.

Bước tiếp theo tốt là triển khai API pastebin được đề cập trong hướng dẫn DRF chính thức[111]. Tôi thậm chí đã viết một hướng dẫn cập nhật cho người mới bắt đầu [112] cho nó có các hướng dẫn từng bước.

Các gói của bên thứ ba cũng rất cần thiết cho việc phát triển Django REST Framework cũng như đối với chính Django. Bạn có thể tìm thấy danh sách đầy đủ tại Django Packages 113 hoặc danh sách được tuyển chọn trên repo django[114] tuyệt vời trên Github.

[109] http://www.django-rest-framework.org/
[110] http://www.django-rest-framework.org/api-guide/testing/
[111] http://www.django-rest-framework.org/tutorial/1-serialization/
[112] https://learndjango.com/tutorials/official-django-rest-framework-tutorial-beginners
[113] https://djangopackages.org/
[114] https://github.com/wsvincent/awesome-django

Giving Thanks

Trong khi cộng đồng Django khá lớn và dựa vào sự làm việc chăm chỉ của nhiều individu- als, Django REST Framework nhỏ hơn nhiều so với. Ban đầu nó được tạo ra bởi Tom Christie[115], một kỹ sư phần mềm người Anh, hiện đang làm việc toàn thời gian nhờ nguồn tài trợ mã nguồn mở. Ông vẫn lãnh đạo sự phát triển tích cực. Nếu bạn thích làm việc với Django REST Framework, vui lòng cân nhắc dành một chút thời gian để cảm ơn cá nhân anh ấy trên Twitter[116].

Và cảm ơn bạn đã đọc cùng và ủng hộ công việc của tôi. Nếu bạn đã mua sách trên Amazon, vui lòng cân nhắc để lại đánh giá trung thực: chúng tạo ra tác động to lớn đến doanh số bán sách và giúp tôi tiếp tục sản xuất cả sách và nội dung Django miễn phí mà tôi thích làm.

[115] http://www.tomchristie.com/

[116] https://twitter.com/_tomchristie