UNIVERSITÀ DELLA CALABRIA

il Campus per eccellenza

Django REST Framework – Part 1

Mario Alviano

Django

- A mature framework for web development (since 2005)
- Written in Python (so we like it)
- Proved to be solid (many tools, libraries, REST framework)
- We introduce the minimum notions to start using it
- We focus on REST APIs (back-end)

These slides are based on the book
Django for APIs – Build web APIs with Python and Django
by William S. Vincent
http://leanpub.com/djangoforapis
(mainly Chapters 5-9)

Outline

- What is a REST API
- Django projects
- Superuser
- Setup Django REST Framework and documentation
- Apps and models
- Define a REST API
- Refactor with viewsets and routers

Why REST APIs

- Monolithic websites should stay in the past
  - Back-end: database models, URLs and views
  - Front-end: templates of HTML, CSS and JavaScript
  - Why mixing the two aspects?
- Modern websites should separate back-end and front-end
  - Django for back-end, and only for data operations
  - Use the front-end you like
  - Use more than one front-end: browser, Android, iOS

HTTP

HTTP is a request-response protocol
Often used for CRUD functionalities

CRUD	**HTTP verbs**
Create	POST
Read	GET
Update	PUT
Delete	DELETE

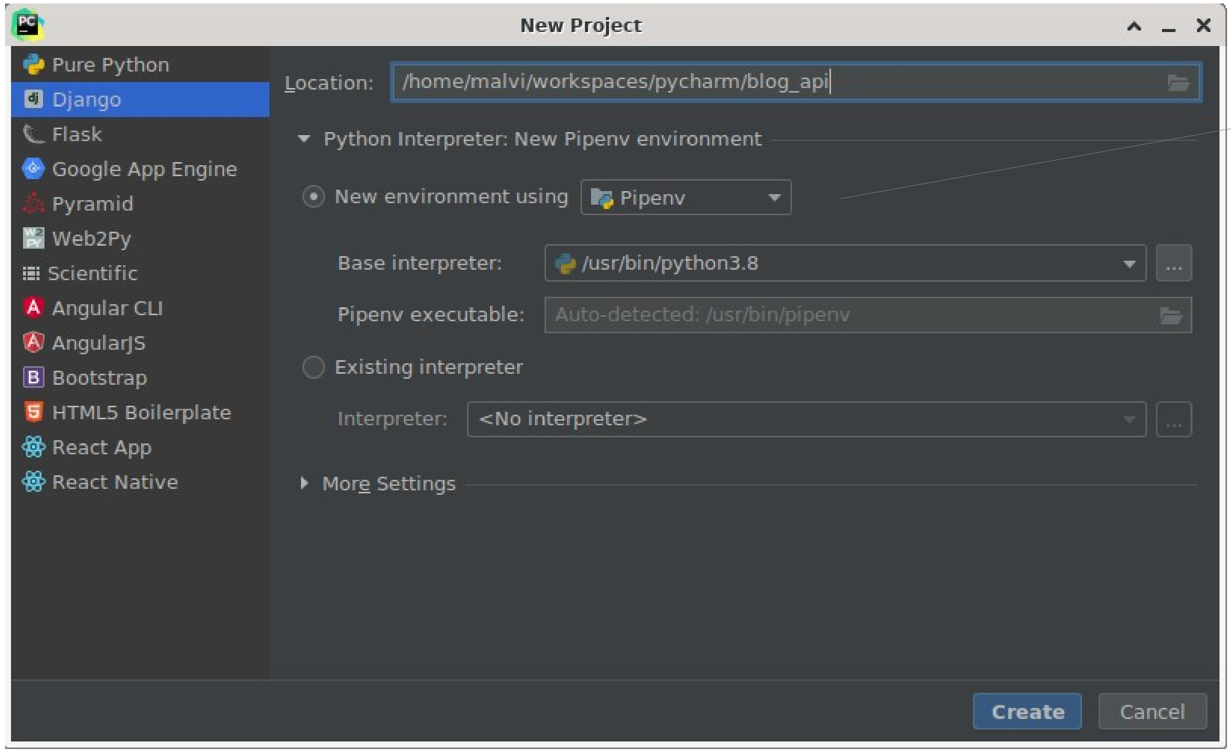
```
https://www.mysite.com/api/users      # GET returns all users

https://www.mysite.com/api/users/<id> # GET returns a single user
```

Endpoints are URLs that expose and
receive data (in JSON or XML)

REST

- REpresentational State Transfer
- Architecture for building APIs on top of HTTP
- Stateless (every request should be independent from previous requests)
- Relies on HTTP verbs (GET, POST, PUT, DELETE, ...)
- Represents data in JSON or XML

Create a new project



New Project

Location: /home/malvi/workspaces/pycharm/blog_api

Python Interpreter: New Pipenv environment

New environment using: Pipenv

Base interpreter: /usr/bin/python3.8

Pipenv executable: Auto-detected: /usr/bin/pipenv

Existing interpreter

Interpreter: <No interpreter>

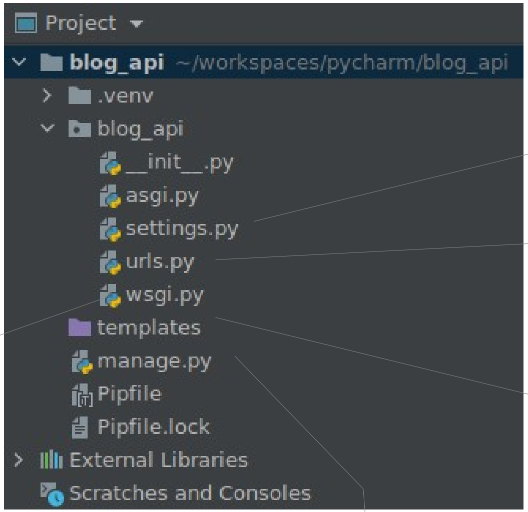
Let’s use pip

The book describes the command-line procedure:

1) create a virtual environment
2) install django
3) create a new project with django-admin startproject blog_api

PyCharm will do all these stuff for us

Anatomy of Django Projects



```
Project
blog_api ~/workspaces/pycharm/blog_api
  .venv
  blog_api
    __init__.py
    asgi.py
    settings.py
    urls.py
    wsgi.py
  templates
  manage.py
  Pipfile
  Pipfile.lock
External Libraries
Scratches and Consoles
```

settings.py contains the configuration of the project

urls.py will contain all routes of the project

templates will contain all HTML pages of the project

WSGI is a standard for Python web servers

ASGI is a standard for asynchronous servers

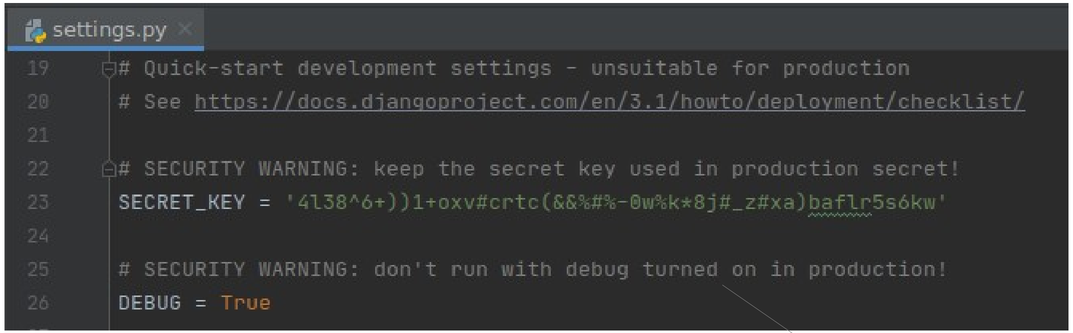
manage.py is a script for the developer to run various Django commands

We will use it, but we usually don’t need to modify it

About settings.py

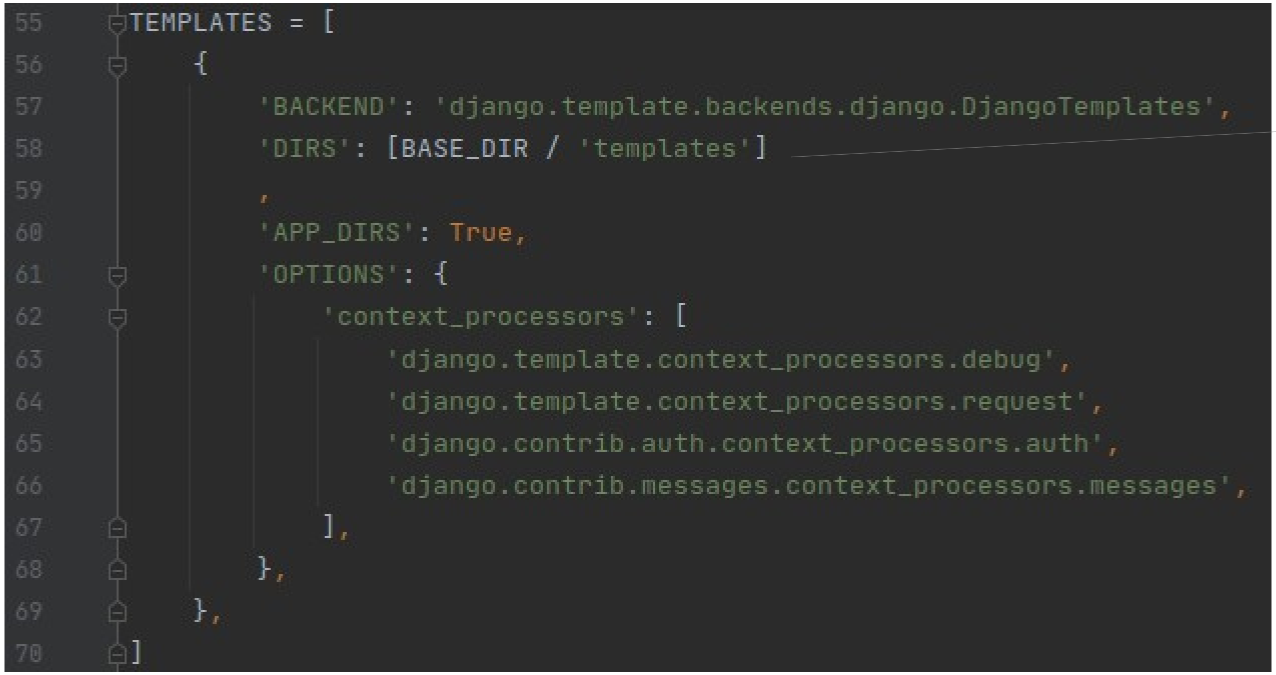
Essentially a file of variable declarations

All variable names are UPPERCASE and considered constants


settings.py

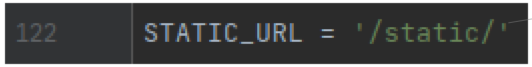
```python
# Quick-start development settings - unsuitable for production
# See https://docs.djangoproject.com/en/3.1/howto/deployment/checklist/

# SECURITY WARNING: keep the secret key used in production secret!
SECRET_KEY = '4l38^6+))1+oxv#crtc(&&%#%-0w%k*8j#_z#xa)baflr5s6kw'

# SECURITY WARNING: don't run with debug turned on in production!
DEBUG = True
```

Before deploying to production,
we have to change something here!



```
TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [BASE_DIR / 'templates']
        ,
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
                'django.contrib.messages.context_processors.messages',
            ],
        },
    },
]
```

Django will search for templates here



```
STATIC_URL = '/static/'
```

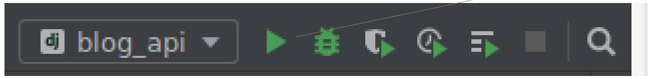
Static resources (aka as-it-is files) are searched here

For REST APIs we don’t really need them!

Remove templates if you want

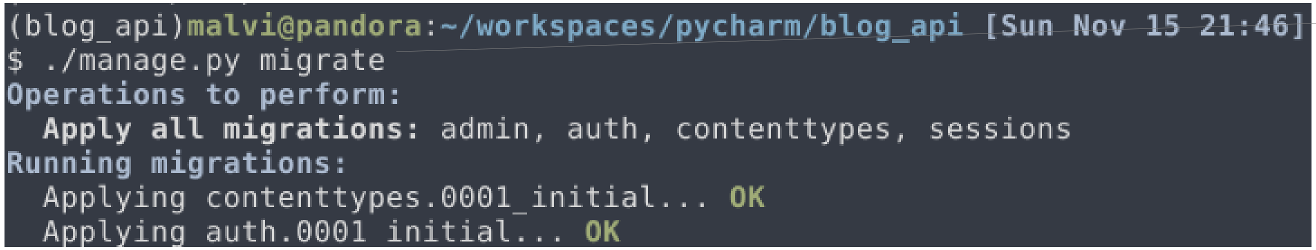
Keep static files for the admin site

Start the project



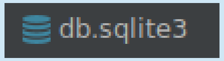
With PyCharm is quite easy:
just click the run button

You should also migrate the database first

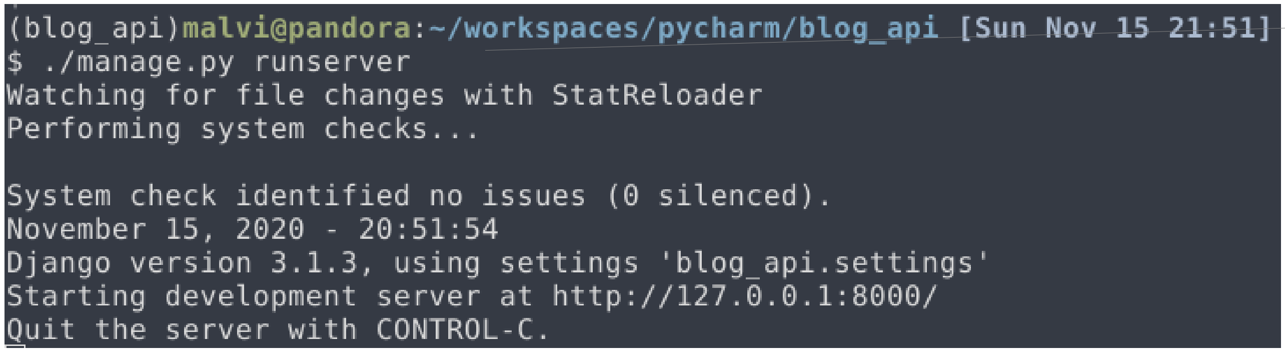


```
(blog_api)malvi@pandora:~/workspaces/pycharm/blog_api [Sun Nov 15 21:46]
$ ./manage.py migrate
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, sessions
Running migrations:
  Applying contenttypes.0001_initial... OK
  Applying auth.0001_initial... OK
```

Create or upgrade
the database



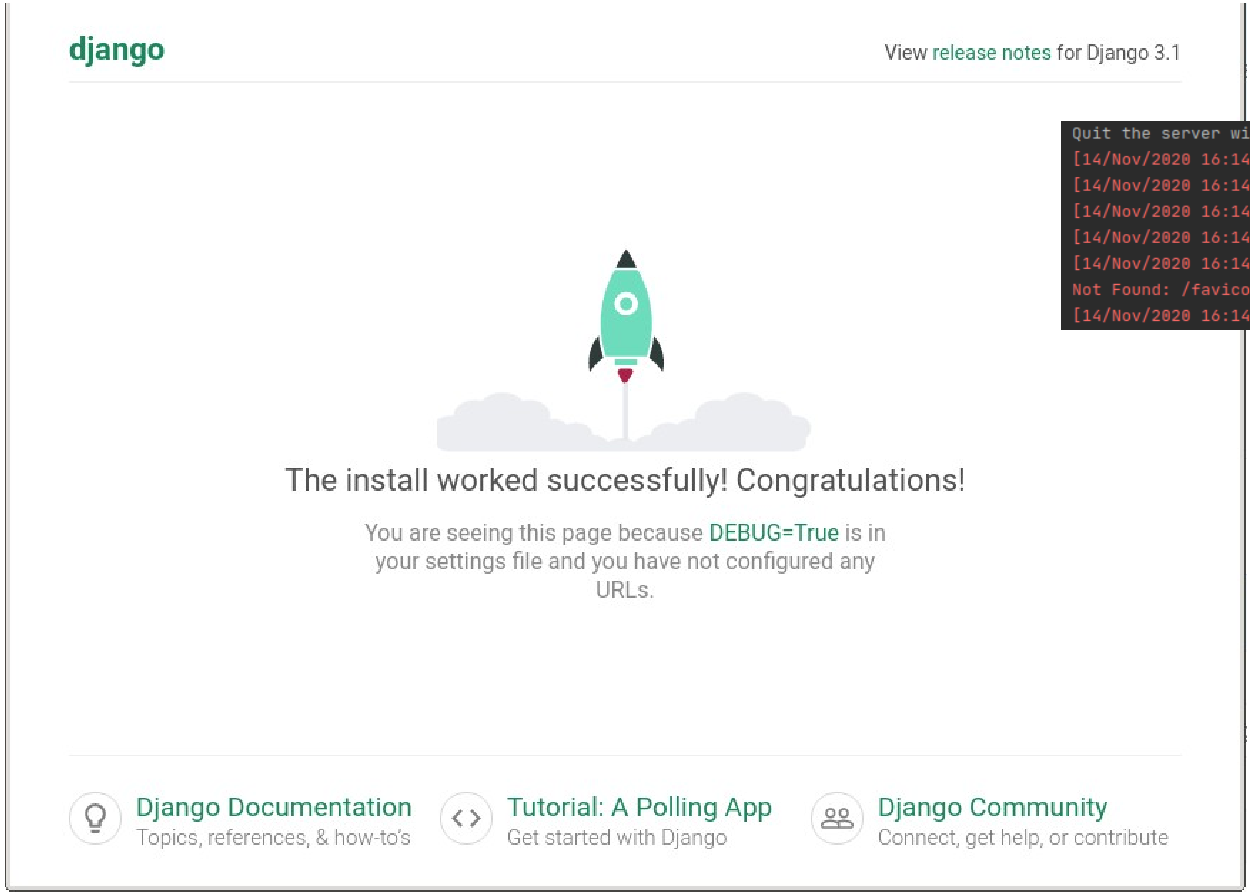
From the command-line (and behind the scene)



```
(blog_api)malvi@pandora:~/workspaces/pycharm/blog_api [Sun Nov 15 21:51]
$ ./manage.py runserver
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).
November 15, 2020 - 20:51:54
Django version 3.1.3, using settings 'blog_api.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CONTROL-C.
```

Start the server
on localhost

Visit http://127.0.0.1:8000/ with your browser


django

View release notes for Django 3.1

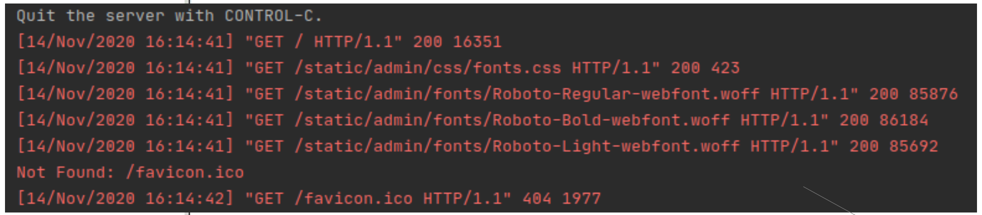
The install worked successfully! Congratulations!

You are seeing this page because DEBUG=True is in your settings file and you have not configured any URLs.

Django Documentation
Topics, references, & how-to's

Tutorial: A Polling App
Get started with Django

Django Community
Connect, get help, or contribute

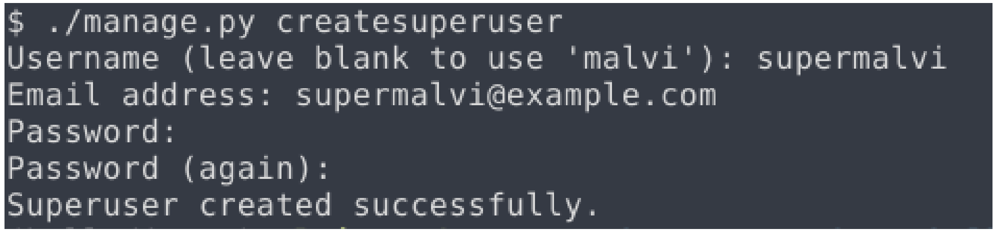

```
Quit the server with CONTROL-C.
[14/Nov/2020 16:14:41] "GET / HTTP/1.1" 200 16351
[14/Nov/2020 16:14:41] "GET /static/admin/css/fonts.css HTTP/1.1" 200 423
[14/Nov/2020 16:14:41] "GET /static/admin/fonts/Roboto-Regular-webfont.woff HTTP/1.1" 200 85876
[14/Nov/2020 16:14:41] "GET /static/admin/fonts/Roboto-Bold-webfont.woff HTTP/1.1" 200 86184
[14/Nov/2020 16:14:41] "GET /static/admin/fonts/Roboto-Light-webfont.woff HTTP/1.1" 200 85692
Not Found: /favicon.ico
[14/Nov/2020 16:14:42] "GET /favicon.ico HTTP/1.1" 404 1977
```

Log and debug information on STDERR

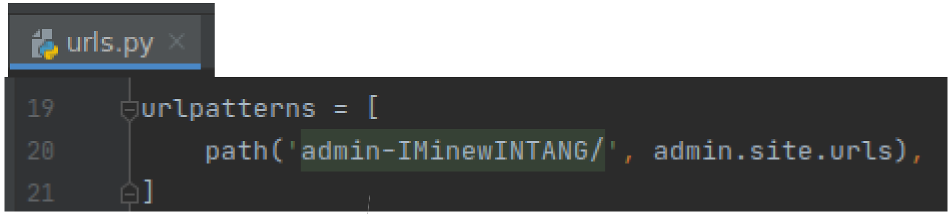
Create a superuser

- Create a superuser from the command-line
- Avoid the username admin (always)
- Password? Generate them

  https://www.lastpass.com/it/password-generator



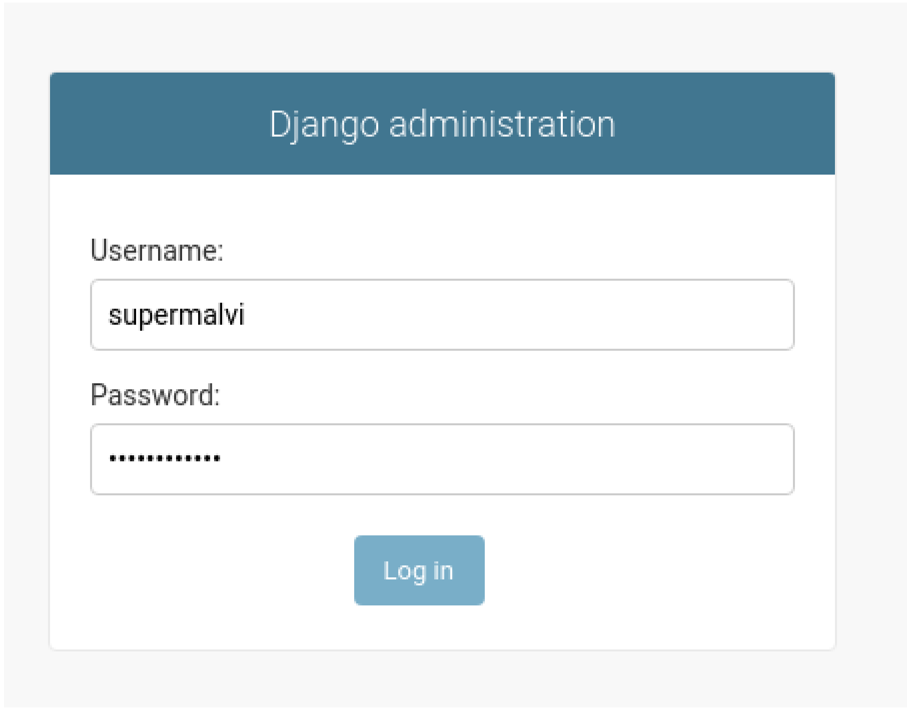
```
$ ./manage.py createsuperuser
Username (leave blank to use 'malvi'): supermalvi
Email address: supermalvi@example.com
Password:
Password (again):
Superuser created successfully.
```



urls.py

```
urlpatterns = [
    path('admin-IMinewINTANG/', admin.site.urls),
]
```

Change the default URL
for the admin site


Django administration

Username:

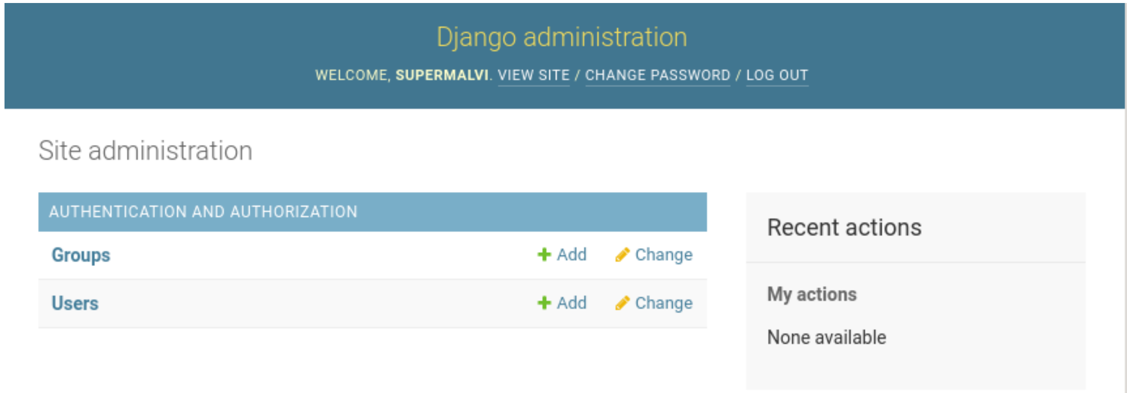
supermalvi

Password:

••••••••••••

Run the server and visit http://127.0.0.1:8000/admin-IMinewINTANG

Log in with the superuser credentials


Django administration

WELCOME, **SUPERMALVI**. VIEW SITE / CHANGE PASSWORD / LOG OUT

Site administration

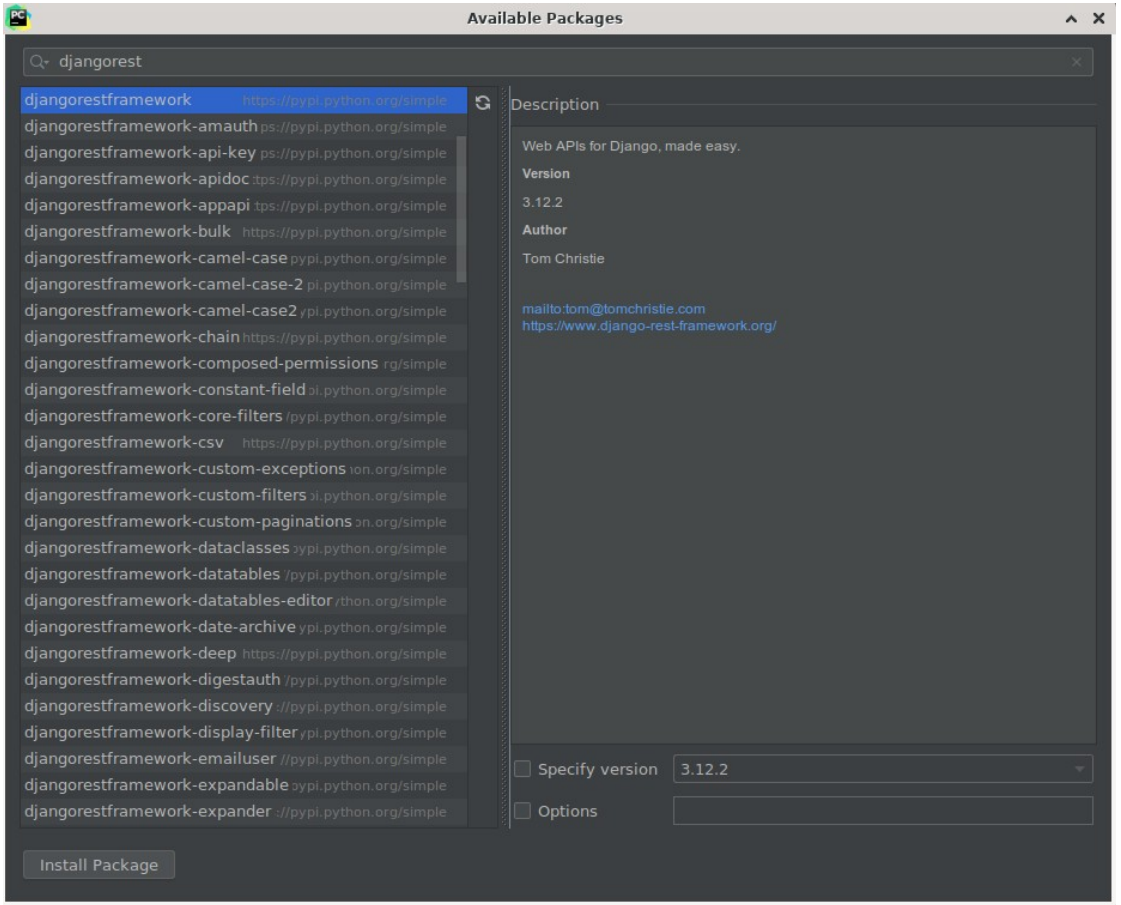
AUTHENTICATION AND AUTHORIZATION		
Groups	Add	Change
Users	Add	Change

Recent actions

My actions

None available

Install Django REST Framework



djangorest

djangorestframework	https://pypi.python.org/simple
djangorestframework-amauth	ps://pypi.python.org/simple
djangorestframework-api-key	ps://pypi.python.org/simple
djangorestframework-apidoc	tps://pypi.python.org/simple
djangorestframework-appapi	tps://pypi.python.org/simple
djangorestframework-bulk	https://pypi.python.org/simple
djangorestframework-camel-case	pypi.python.org/simple
djangorestframework-camel-case-2	pi.python.org/simple
djangorestframework-camel-case2	ypi.python.org/simple
djangorestframework-chain	https://pypi.python.org/simple
djangorestframework-composed-permissions	rg/simple
djangorestframework-constant-field	pi.python.org/simple
djangorestframework-core-filters	/pypi.python.org/simple
djangorestframework-csv	https://pypi.python.org/simple
djangorestframework-custom-exceptions	non.org/simple
djangorestframework-custom-filters	pi.python.org/simple
djangorestframework-custom-paginations	on.org/simple
djangorestframework-dataclasses	pypi.python.org/simple
djangorestframework-datatables	/pypi.python.org/simple
djangorestframework-datatables-editor	thon.org/simple
djangorestframework-date-archive	ypi.python.org/simple
djangorestframework-deep	https://pypi.python.org/simple
djangorestframework-digestauth	/pypi.python.org/simple
djangorestframework-discovery	://pypi.python.org/simple
djangorestframework-display-filter	ypi.python.org/simple
djangorestframework-emailuser	//pypi.python.org/simple
djangorestframework-expandable	pypi.python.org/simple
djangorestframework-expander	://pypi.python.org/simple

Description

Web APIs for Django, made easy.

Version

3.12.2

Author

Tom Christie

mailto:tom@tomchristie.com
https://www.django-rest-framework.org/

Specify version: 3.12.2

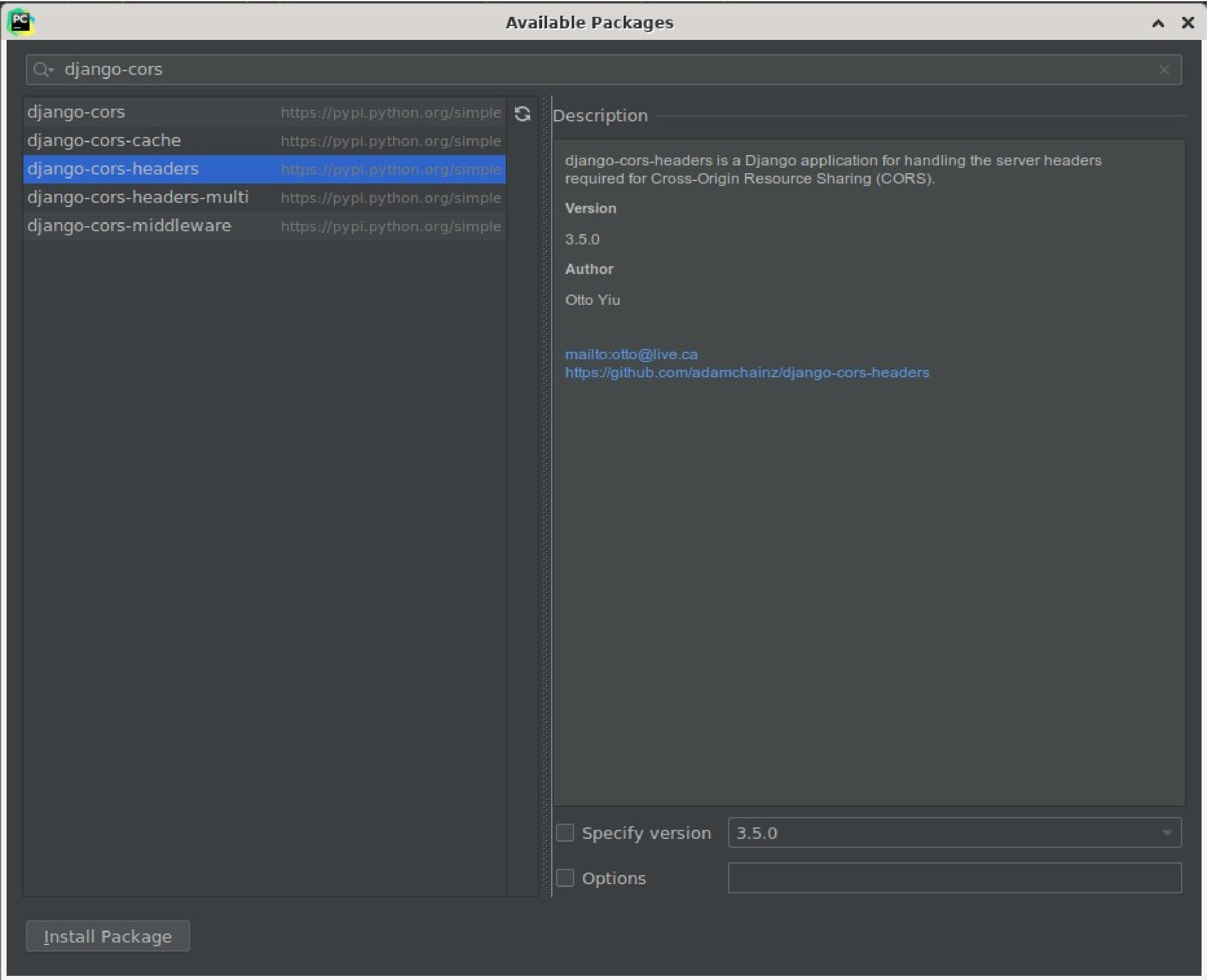
Options:

Menu File | Settings

Project | Python Interpreter

Add the package djangorestframework

Pipfile is updated!


Available Packages
django-cors
django-cors
https://pypi.python.org/simple
django-cors-cache
https://pypi.python.org/simple
django-cors-headers
https://pypi.python.org/simple
django-cors-headers-multi
https://pypi.python.org/simple
django-cors-middleware
https://pypi.python.org/simple
Description
Specify version
Options
Install Package

django-cors-headers is a Django application for handling the server headers required for Cross-Origin Resource Sharing (CORS).

Version

3.5.0

Author

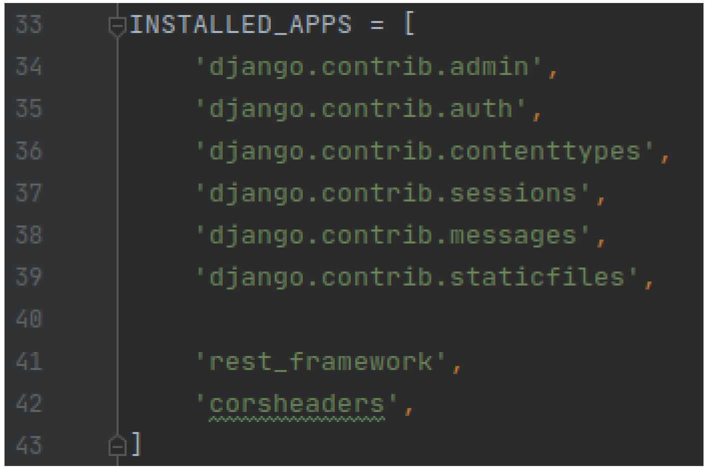
Otto Yiu

mailto:otto@live.ca
https://github.com/adamchainz/django-cors-headers

Specify version: 3.5.0

Since you are there, install also django-cors-headers

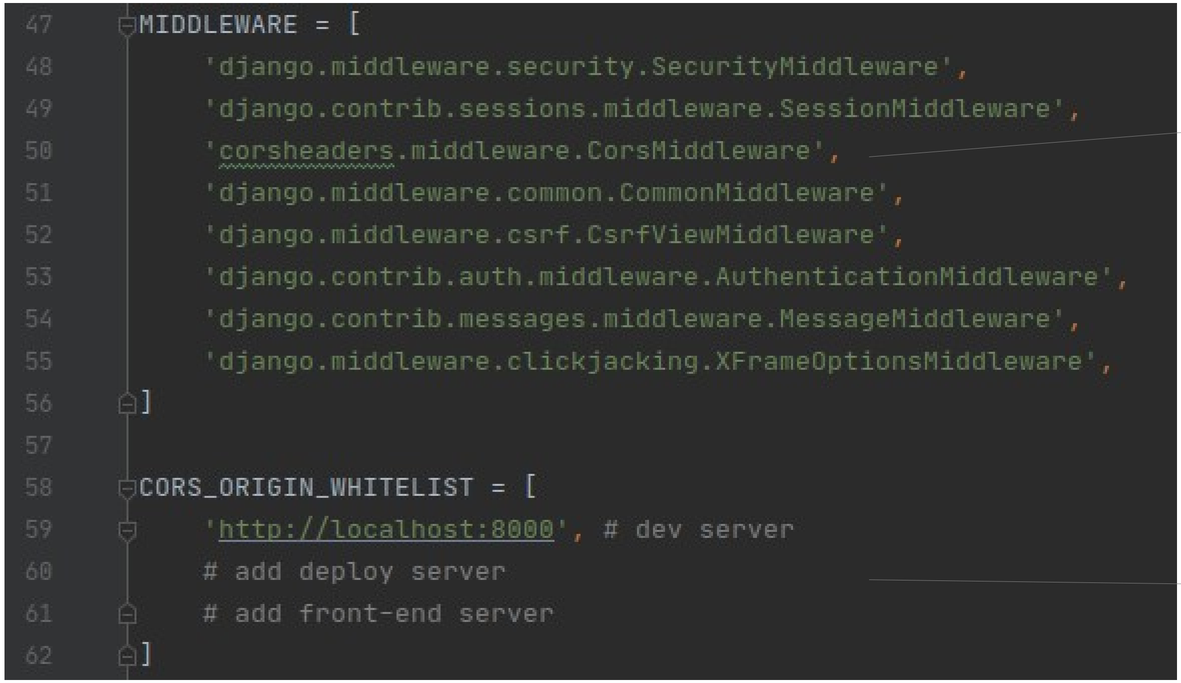
The front-end will be on a different server, so we want to limit requests to known domains



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

    'rest_framework',
    'corsheaders',
]
```

Add rest_framework and corsheaders to the installed apps in settings.py

We will set permissions later



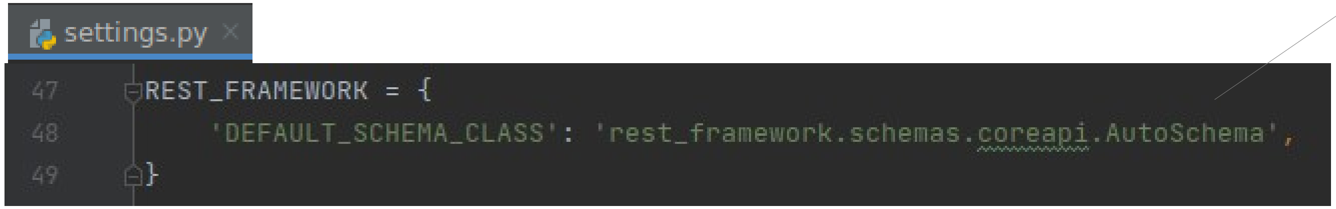
```
MIDDLEWARE = [
    'django.middleware.security.SecurityMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
    'corsheaders.middleware.CorsMiddleware',
    'django.middleware.common.CommonMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',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

CORS_ORIGIN_WHITELIST = [
    'http://localhost:8000', # dev server
    # add deploy server
    # add front-end server
]
```

Add CorsMiddleware before CommonMiddleware

Whitelist allowed origins

Let’s also setup the documentation

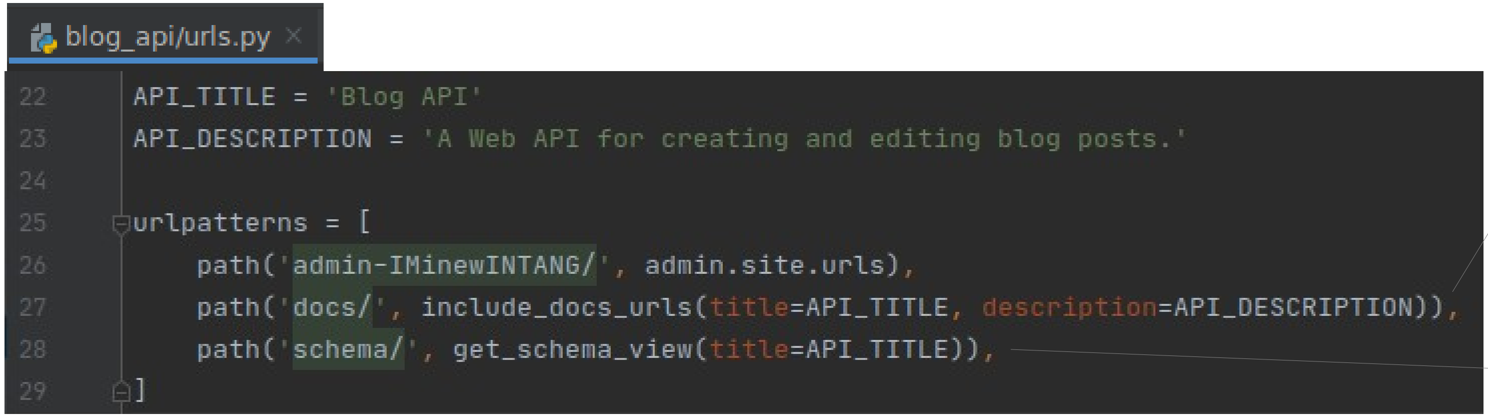
Install coreapi and pyyaml



settings.py

```
REST_FRAMEWORK = {
    'DEFAULT_SCHEMA_CLASS': 'rest_framework.schemas.coreapi.AutoSchema',
}
```

Add this dictionary in settings.py



blog_api/urls.py

```
API_TITLE = 'Blog API'
API_DESCRIPTION = 'A Web API for creating and editing blog posts.'

urlpatterns = [
    path('admin-IMinewINTANG/', admin.site.urls),
    path('docs/', include_docs_urls(title=API_TITLE, description=API_DESCRIPTION)),
    path('schema/', get_schema_view(title=API_TITLE)),
]
```

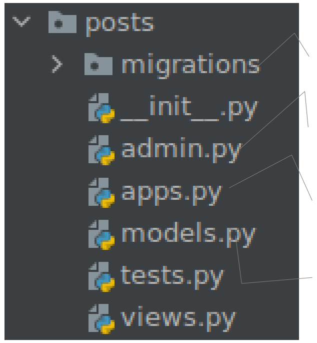
Human-readable documentation

Machine-readable documentation

We will check these URLs later

Creating Apps (isolated components)

- Run **./manage.py startapp posts**
- A new module is added

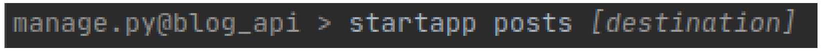


Stores migration files to upgrade the database

Add content to the admin site

App specific configuration

Database model, tests and views



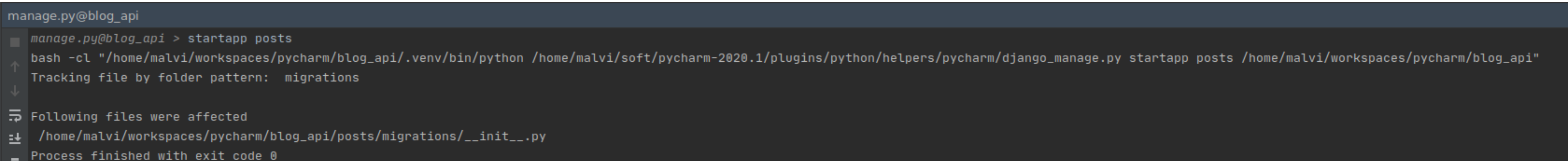
```
manage.py@blog_api > startapp posts [destination]
```

<u>Alternative</u>

Menu **Tools | Run manage.py Task…**

Type **startapp posts**

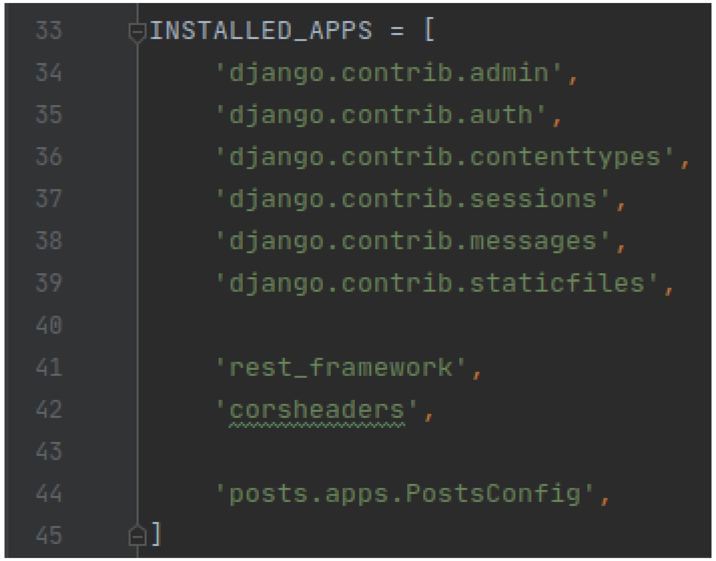
PyCharm provides autocompletion



```
manage.py@blog_api
manage.py@blog_api > startapp posts
bash -cl "/home/malvi/workspaces/pycharm/blog_api/.venv/bin/python /home/malvi/soft/pycharm-2020.1/plugins/python/helpers/pycharm/django_manage.py startapp posts /home/malvi/workspaces/pycharm/blog_api"
Tracking file by folder pattern:  migrations

Following files were affected
 /home/malvi/workspaces/pycharm/blog_api/posts/migrations/__init__.py
Process finished with exit code 0
```

Install the app

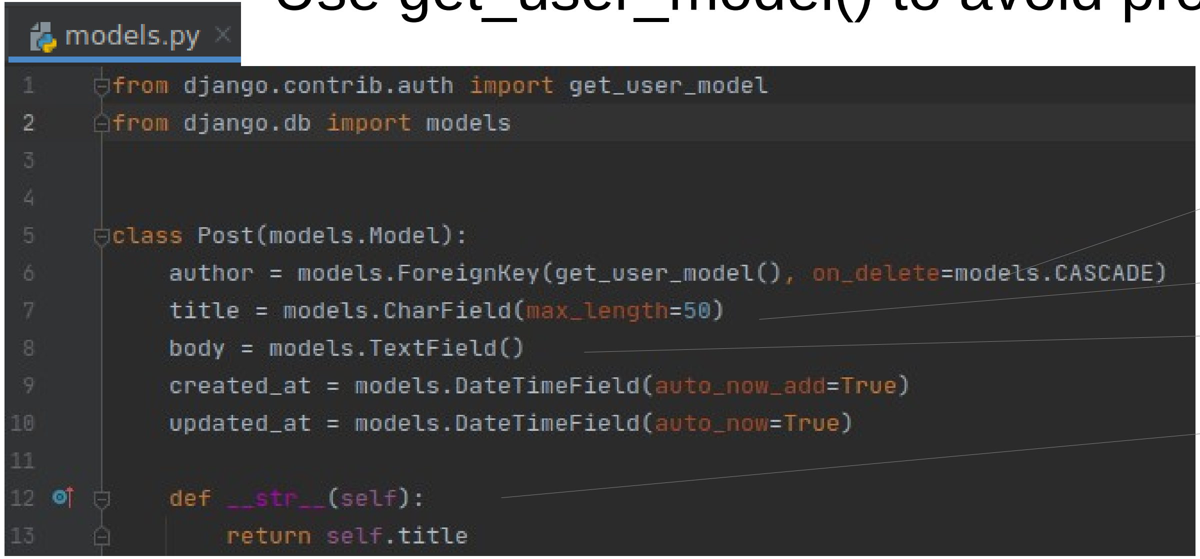
- Add the app to settings.py



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

    'rest_framework',
    'corsheaders',

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

Define the model

- We want a Post table with five fields: author, title, body, created_at, updated_at
- Django provides a User model (aka table)
  - Use get_user_model() to avoid problems


models.py

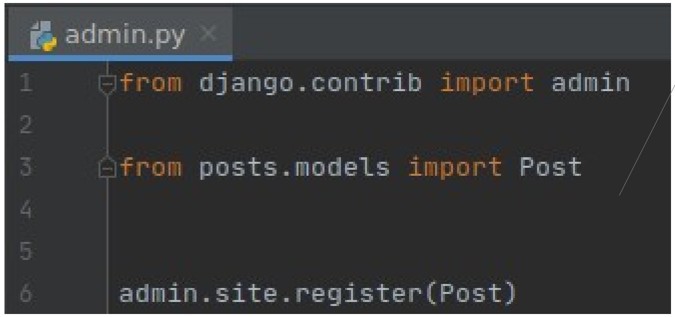
```python
from django.contrib.auth import get_user_model
from django.db import models


class Post(models.Model):
    author = models.ForeignKey(get_user_model(), 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
```

Join to other tables

Small strings

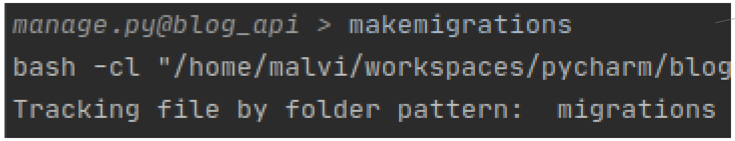
Large amounts of text

Used by the admin site



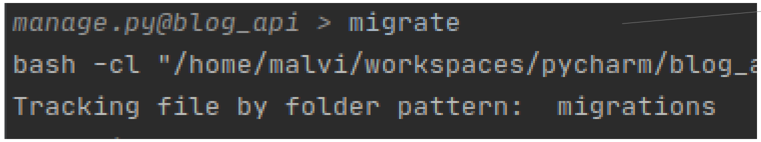
```python
from django.contrib import admin

from posts.models import Post


admin.site.register(Post)
```

Let’s add Post to the admin site



```
manage.py@blog_api > makemigrations
bash -cl "/home/malvi/workspaces/pycharm/blog
Tracking file by folder pattern:  migrations
```

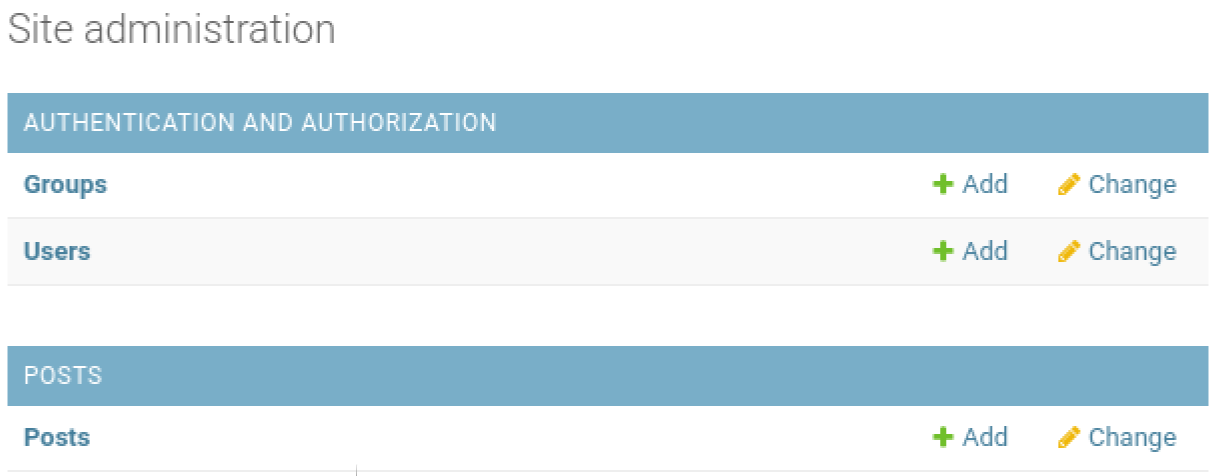
Make migration files
(you may want to put them in git)



```
manage.py@blog_api > migrate
bash -cl "/home/malvi/workspaces/pycharm/blog_a
Tracking file by folder pattern:  migrations
```

Upgrade the database

Visit the admin site


Site administration

AUTHENTICATION AND AUTHORIZATION		
Groups	Add	Change
Users	Add	Change

POSTS		
Posts	Add	Change

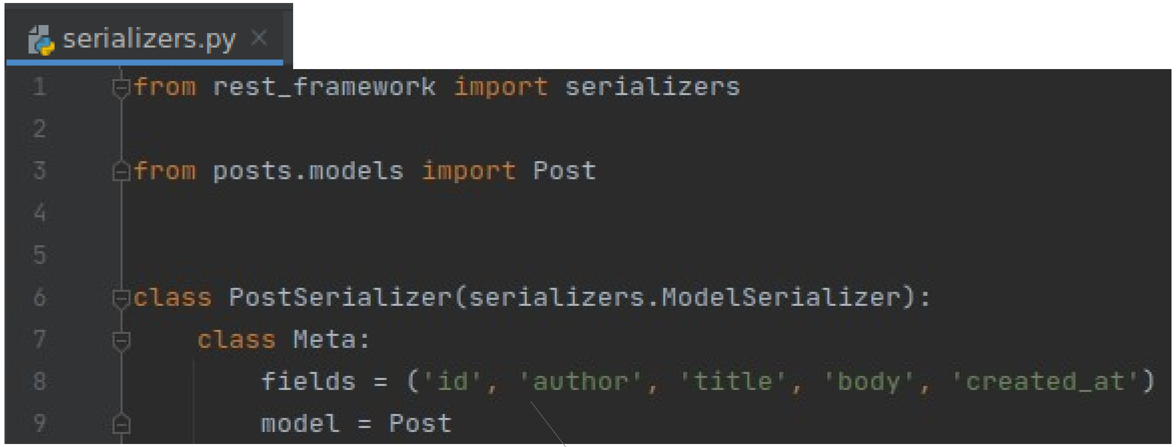
The app POSTS has the objects Posts

Let’s add a couple of posts, try to do mistakes, and so on

Define the REST API

- Three main steps
  - add serializers.py to produce JSON
  - use views.py to apply logic to each API endpoint
  - add urls.py for URL routes

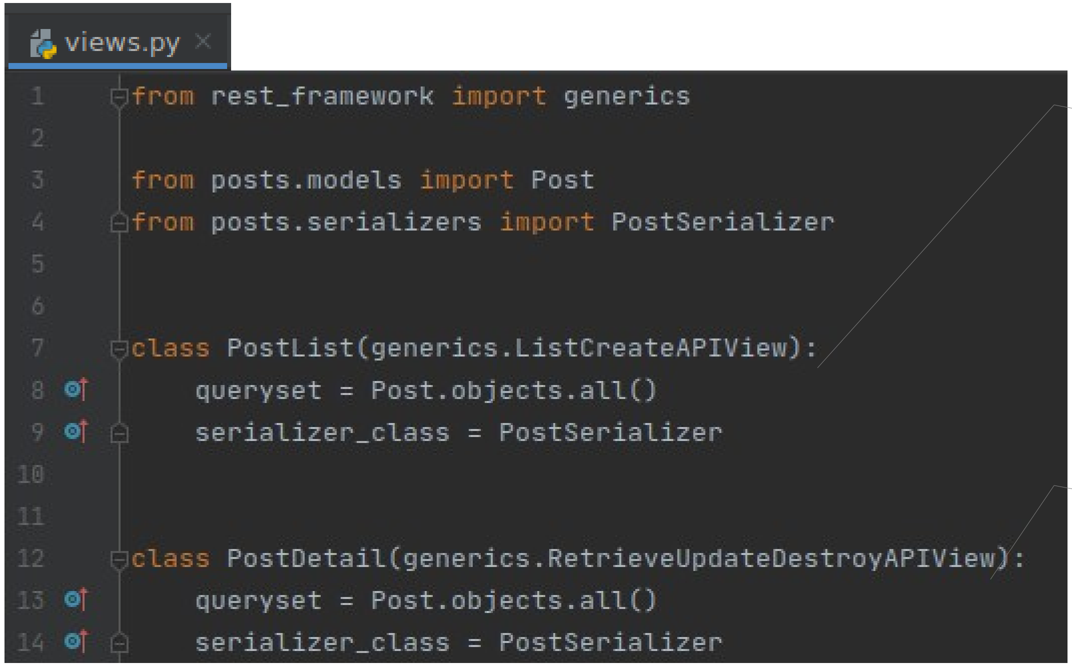
Add serializers.py to the app directory


serializers.py

```python
from rest_framework import serializers

from posts.models import Post


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

With ModelSerializer is easy as specifing the model and the fields to expose

Modify views.py

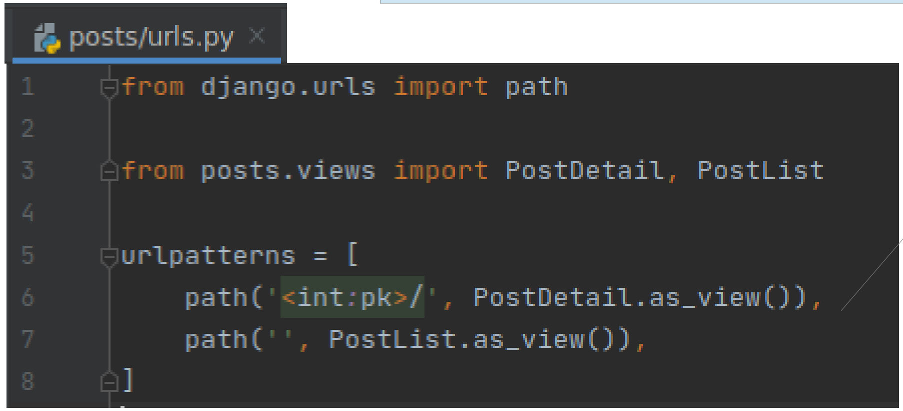


```python
from rest_framework import generics

from posts.models import Post
from posts.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
```

List all posts

All operations for a single post

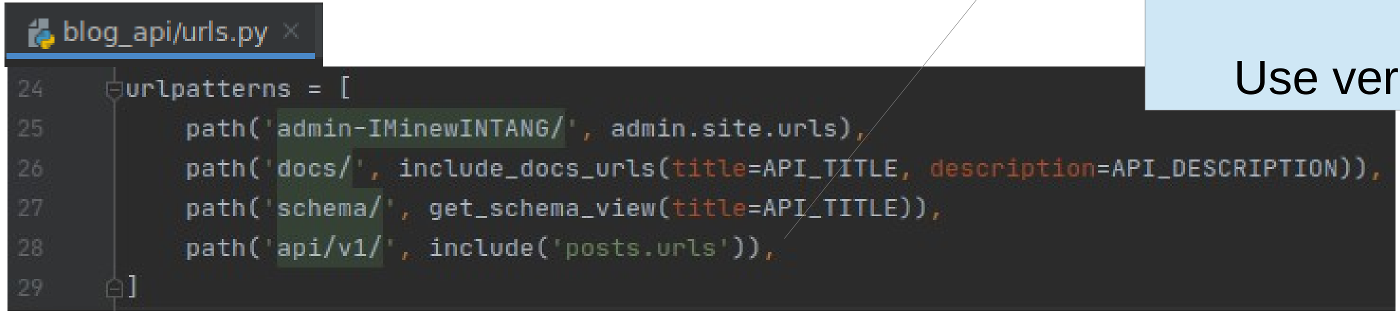
Add urls.py to the app directory


posts/urls.py

```python
from django.urls import path

from posts.views import PostDetail, PostList

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

Empty path to list all posts

Primary key to operate on a post


blog_api/urls.py

```python
urlpatterns = [
    path('admin-IMinewINTANG/', admin.site.urls),
    path('docs/', include_docs_urls(title=API_TITLE, description=API_DESCRIPTION)),
    path('schema/', get_schema_view(title=API_TITLE)),
    path('api/v1/', include('posts.urls')),
]
```

Include the new file in the main urls.py

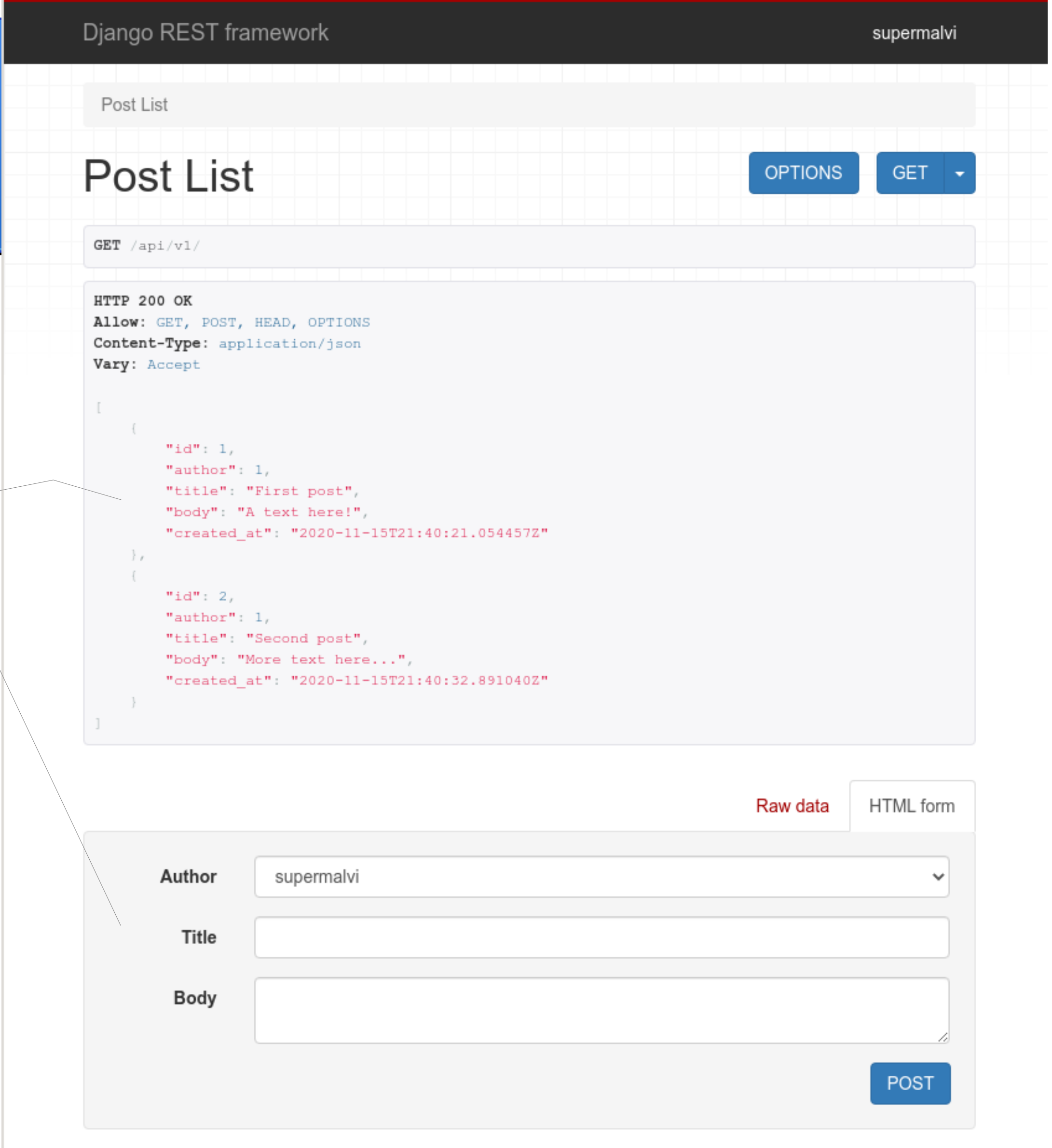
Use version number for the URL

Browsable API

Visit http://127.0.0.1:8000/api/v1/

List of posts here

A new record can be added here



Post List

```
GET /api/v1/
```

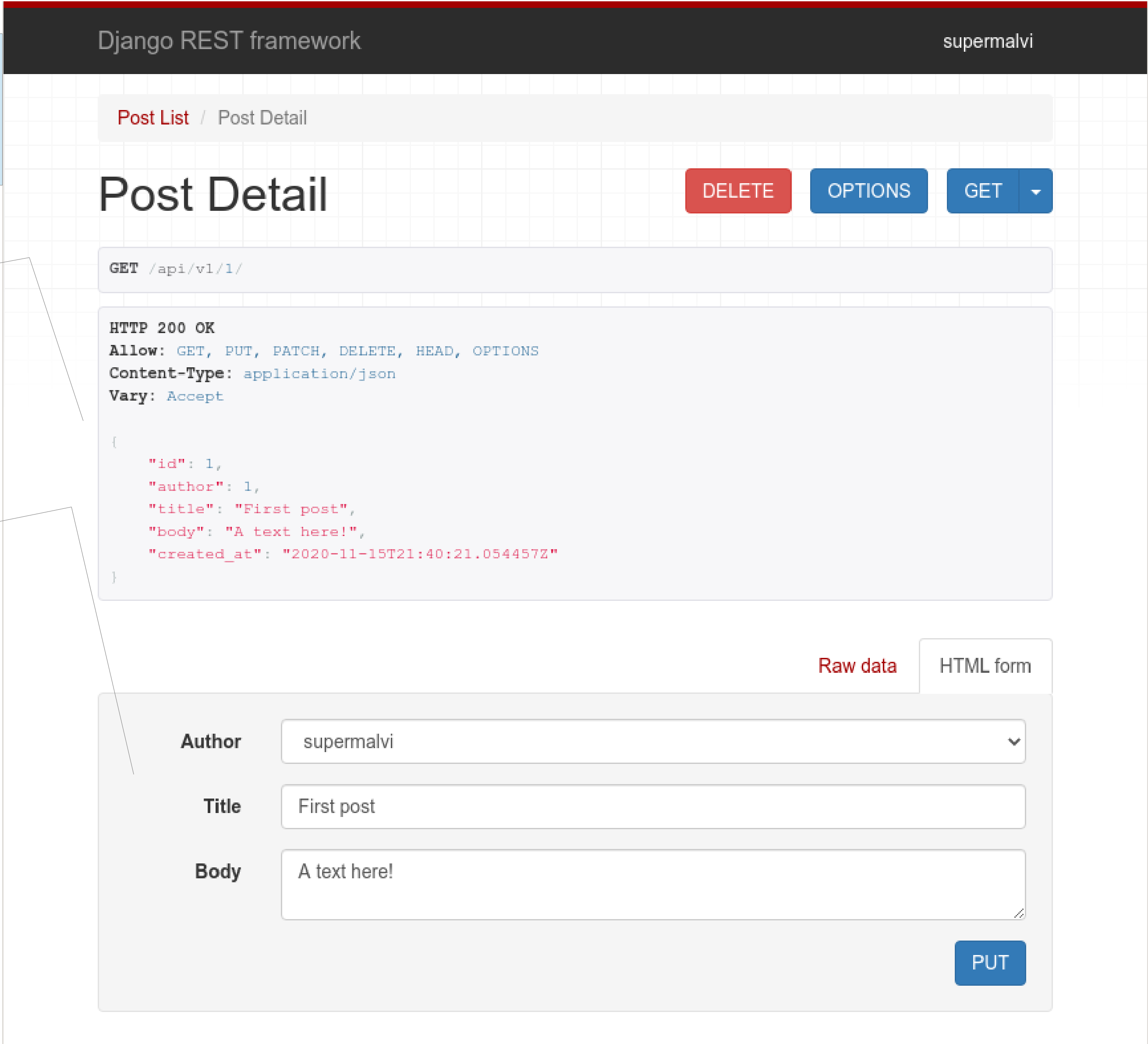
```
HTTP 200 OK
Allow: GET, POST, HEAD, OPTIONS
Content-Type: application/json
Vary: Accept

[
    {
        "id": 1,
        "author": 1,
        "title": "First post",
        "body": "A text here!",
        "created_at": "2020-11-15T21:40:21.054457Z"
    },
    {
        "id": 2,
        "author": 1,
        "title": "Second post",
        "body": "More text here...",
        "created_at": "2020-11-15T21:40:32.891040Z"
    }
]
```

Author	supermalvi
Title	
Body	

Visit http://127.0.0.1:8000/api/v1/1/

Post details here

The record can be updated here

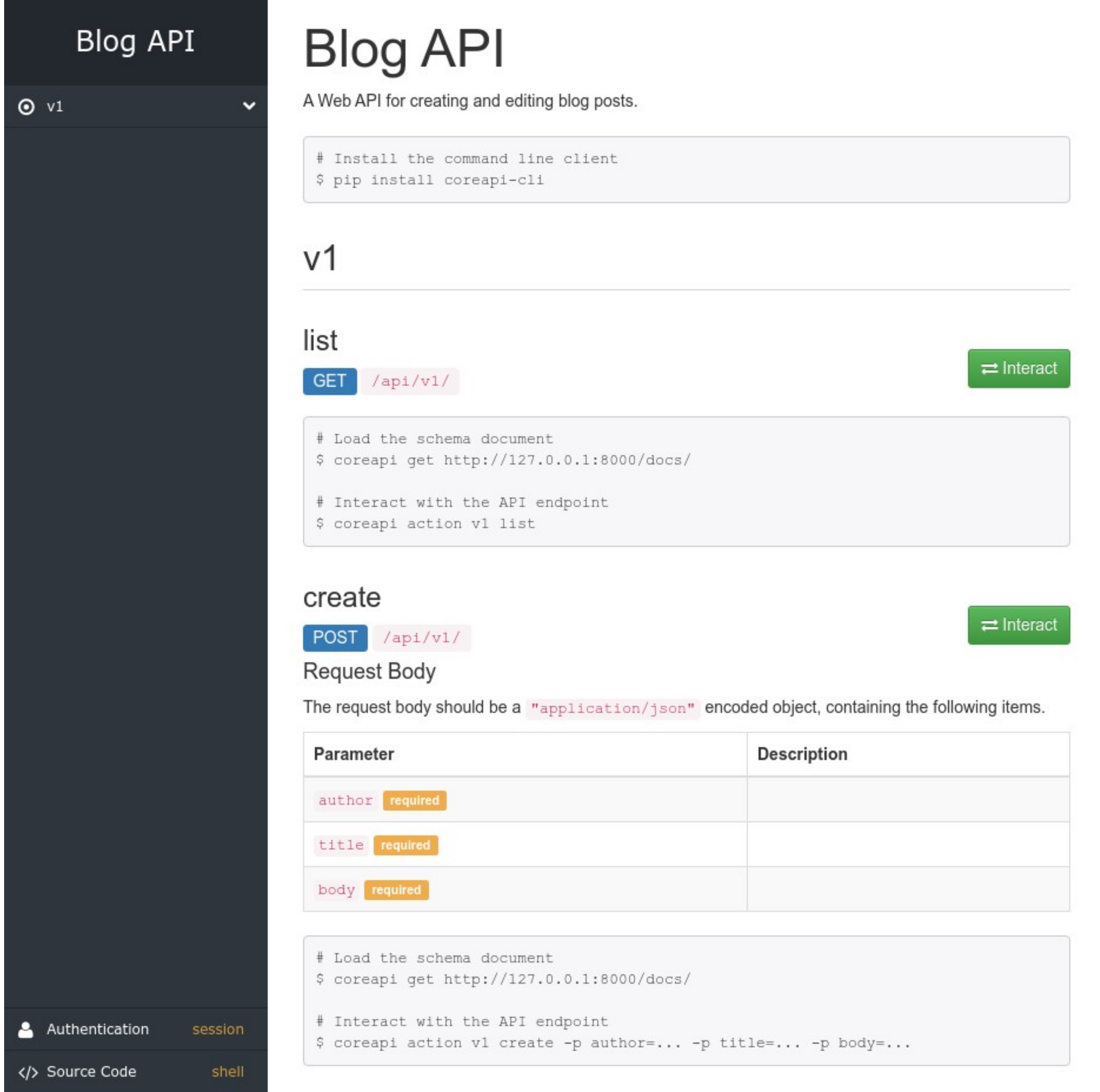


```
GET /api/v1/1/
```

```
HTTP 200 OK
Allow: GET, PUT, PATCH, DELETE, HEAD, OPTIONS
Content-Type: application/json
Vary: Accept

{
    "id": 1,
    "author": 1,
    "title": "First post",
    "body": "A text here!",
    "created_at": "2020-11-15T21:40:21.054457Z"
}
```

Author	supermalvi
Title	First post
Body	A text here!

Visit http://127.0.0.1:8000/docs/



Blog API

A Web API for creating and editing blog posts.

```
# Install the command line client
$ pip install coreapi-cli
```

v1

list

GET /api/v1/

```
# Load the schema document
$ coreapi get http://127.0.0.1:8000/docs/

# Interact with the API endpoint
$ coreapi action v1 list
```

create

POST /api/v1/

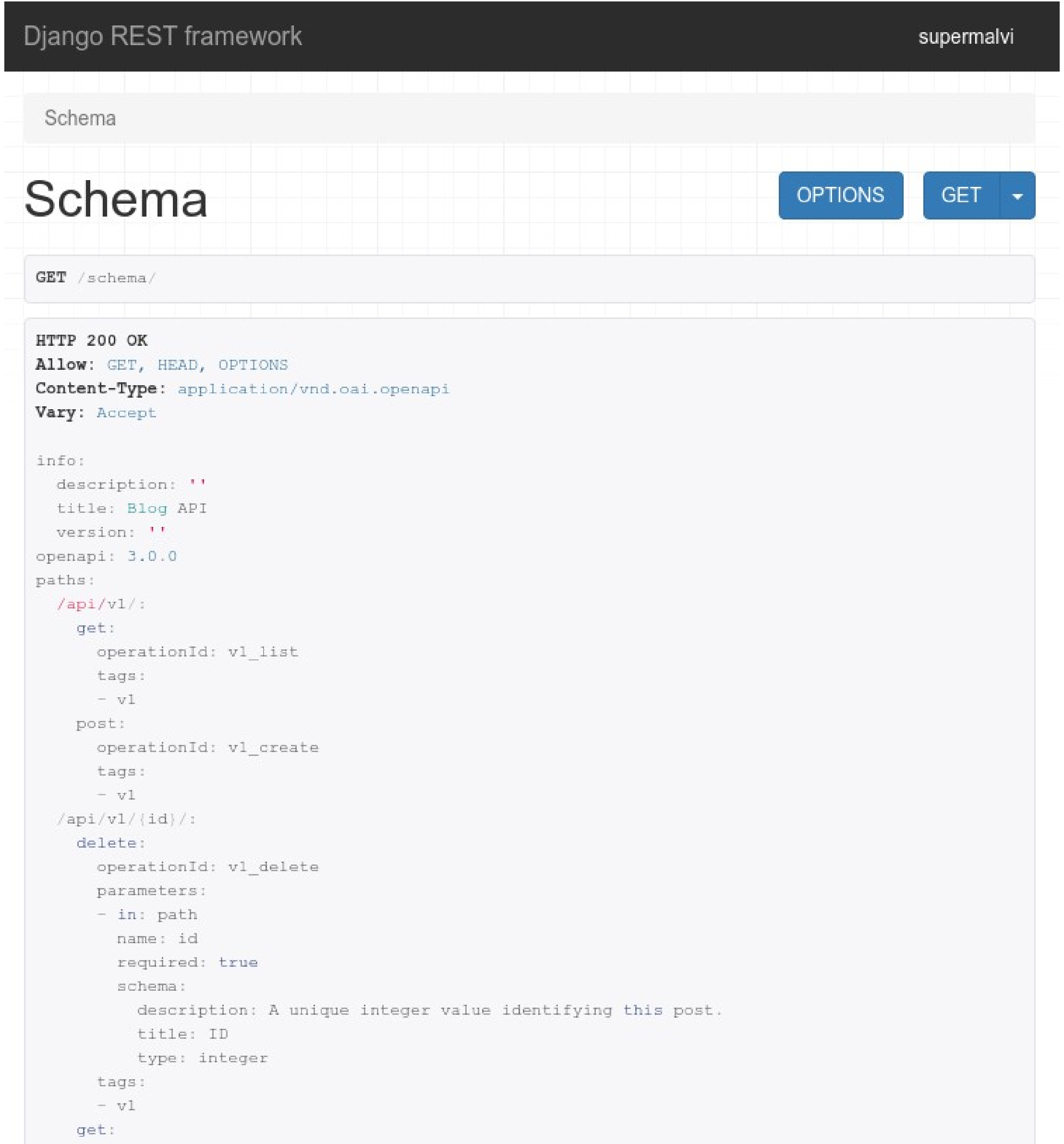
Request Body

The request body should be a `"application/json"` encoded object, containing the following items.

Parameter	Description
author required	
title required	
body required	

```
# Load the schema document
$ coreapi get http://127.0.0.1:8000/docs/

# Interact with the API endpoint
$ coreapi action v1 create -p author=... -p title=... -p body=...
```

Visit http://127.0.0.1:8000/schema/

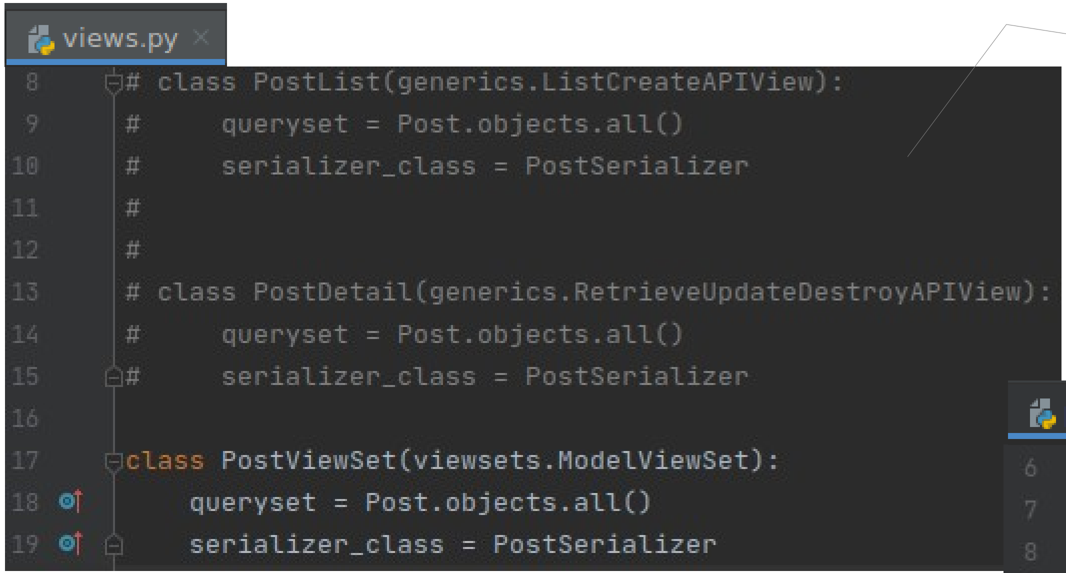


Schema

```
GET /schema/
```

```
HTTP 200 OK
Allow: GET, HEAD, OPTIONS
Content-Type: application/vnd.oai.openapi
Vary: Accept

info:
  description: ''
  title: Blog API
  version: ''
openapi: 3.0.0
paths:
  /api/v1/:
    get:
      operationId: v1_list
      tags:
      - v1
    post:
      operationId: v1_create
      tags:
      - v1
  /api/v1/{id}/:
    delete:
      operationId: v1_delete
      parameters:
      - in: path
        name: id
        required: true
        schema:
          description: A unique integer value identifying this post.
          title: ID
          type: integer
      tags:
      - v1
    get:
```

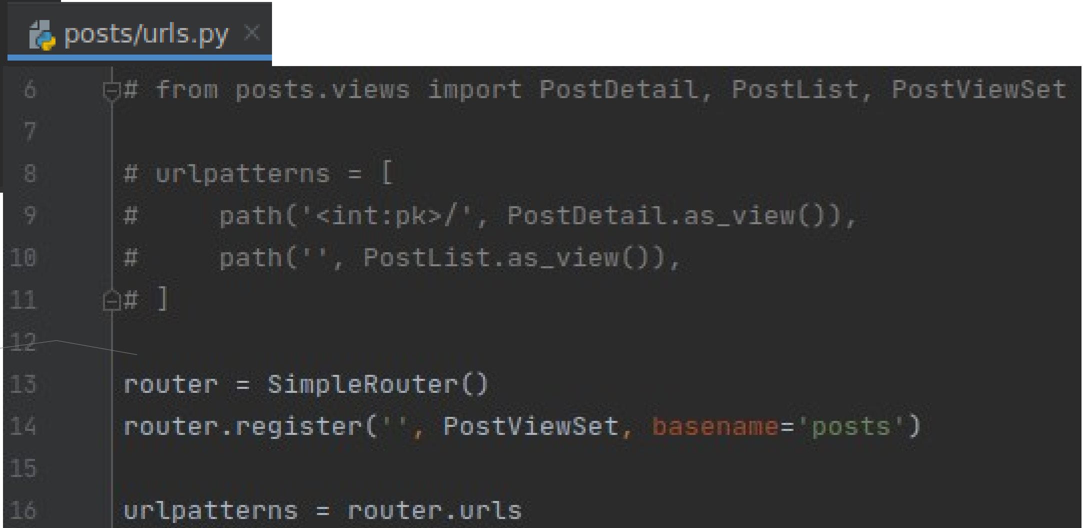
Refactor



views.py

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

class PostViewSet(viewsets.ModelViewSet):
    queryset = Post.objects.all()
    serializer_class = PostSerializer
```

A viewset can replace multiple views



posts/urls.py

```
# from posts.views import PostDetail, PostList, PostViewSet

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

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

urlpatterns = router.urls
```

A router generates URLs for a viewset

Questions