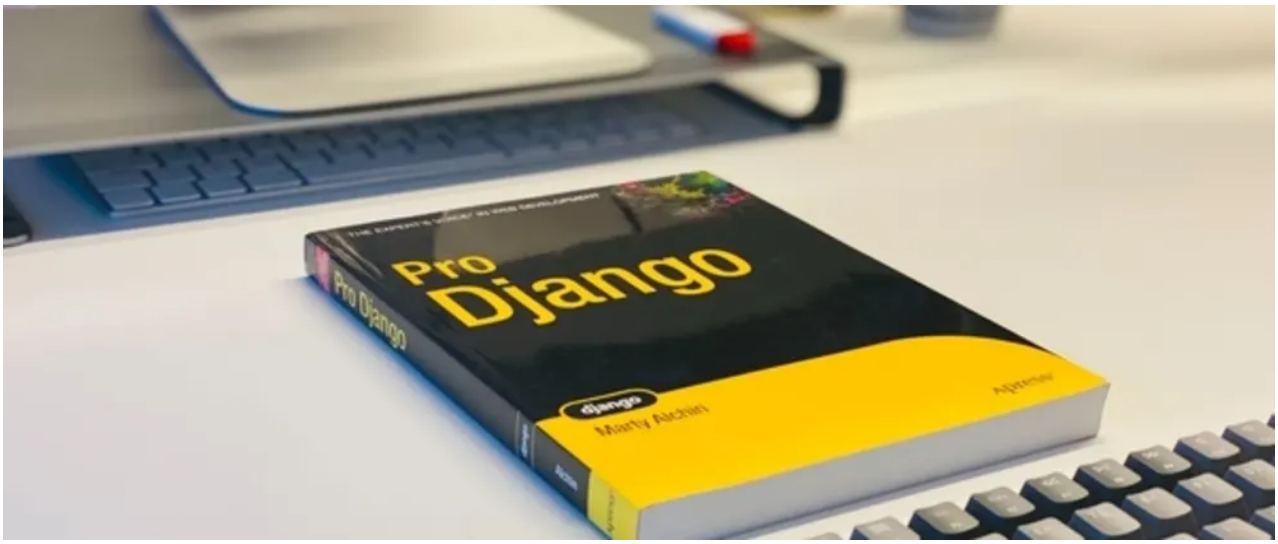
How to create Django REST APIs

dev.to/paulodhiambo/how-to-create-django-rest-apis-150m



In this tutorial, we will create a fully working to-do CRUD Django API using Django and Django rest framework. Restful API endpoints make it possible to perform CRUD functionality in the backend from within the mobile app or website.

Prerequisites

To verify if Python is installed and configured correctly on your system, Open the terminal and type in the command `python3 --version` on Linux/Mac or `python --version` if you are on Windows.

```
$ python3 --version
Python 3.8.5
```

To verify virtualenv installation execute the command `virtualenv --version` on the terminal.

```
$ virtualenv --version
virtualenv 20.0.35 from /home/user/.local/lib/python3.8/site-
packages/virtualenv/__init__.py
```

Creating the Todo project

We will start by creating our project's work directory and a virtual environment for our project. The virtual environment makes it possible to run our project and its dependencies in an isolated environment.

Run `mkdir ~/todo` to create our working directory.

```
$ mkdir ~/todo
$ cd todo
```

1. To create a virtual environment for our project run:

```
$ virtualenv venv
```

`venv` in in the command `virtualenv venv` is the name of our virtual environment

1. To activate the virtual environment for our projected run:

```
$ source venv/bin/activate
```

1. To install Django and Django rest framework in our virtual environment run:

```
$ pip3 install django
```

Running the command will result in something similar to the code block below

```
$ pip3 install django
Collecting django
Using cached Django-3.1.3-py3-none-any.whl (7.8 MB)
Collecting asgiref<4,>=3.2.10
 Downloading asgiref-3.3.1-py3-none-any.whl (19 kB)
Collecting pytz
  Using cached pytz-2020.4-py2.py3-none-any.whl (509 kB)
Collecting sqlparse>=0.2.2
 Using cached sqlparse-0.4.1-py3-none-any.whl (42 kB)
Installing collected packages: asgiref, pytz, sqlparse, django
Successfully installed asgiref-3.3.1 django-3.1.3 pytz-2020.4 sqlparse-0.4.1
```

1. To Django rest framework in our virtual environment run:

```
$ pip3 install djangorestframework
```

Running the command will result in something similar to the code block below:

```
$ pip3 install djangorestframework
Collecting djangorestframework
  Downloading djangorestframework-3.12.2-py3-none-any.whl (957 kB)
     |████████████████████████████████| 957 kB 595 kB/s
Requirement already satisfied: django>=2.2 in ./venv/lib/python3.8/site-
packages (from djangorestframework) (3.1.3)
Requirement already satisfied: asgiref<4,>=3.2.10 in ./venv/lib/python3.8/site-
packages (from django>=2.2->djangorestframework) (3.3.1)
Requirement already satisfied: sqlparse>=0.2.2 in ./venv/lib/python3.8/site-
packages (from django>=2.2->djangorestframework) (0.4.1)
Requirement already satisfied: pytz in ./venv/lib/python3.8/site-packages (from
django>=2.2->djangorestframework) (2020.4)
Installing collected packages: djangorestframework
Successfully installed djangorestframework-3.12.2
```

Let's create our django_todo project now by running the command `django-admin startproject django_todo`.

```
$ django-admin startproject django_todo
```

After successfully creating the project we change our working directory to our project folder and run the Django development server.

```
$ cd django_todo
$ ./manage.py runserver
Watching for file changes with StatReloader
Performing system checks...
System check identified no issues (0 silenced).
You have 18 unapplied migration(s). Your project may not work properly until you
apply the migrations for app(s): admin, auth, contenttypes, sessions.
Run `python manage.py migrate` to apply them.
November 17, 2020 - 10:14:01
Django version 3.1.3, using settings 'django_todo.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CONTROL-C.
```

When we visit `http://127.0.0.1:8000/` we should see the Django welcome page. Django makes it easy for us to organize our code into reusable apps. We are going to create an app for the todo endpoints.
To create an app, run:

```
$ python3 manage.py startapp todo
```

The command above creates a directory named `todo` and generates boilerplate code for a todo app. Now we can plug our todo app into our `django_todo` project.
Open `settings.py` in the project directory `django_todo` and add our `todo` app to the `INSTALLED_APPS` list. We will also add the `rest_framework` app to make it available for use in our project.

```
# ./django_todo/settings.py
...
INSTALLED_APPS = [
    ...
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'todo',
    'rest_framework',
]
```

Creating the Todo model

In Django, models are python classes that represent a table in the database. The Todo Model represents a Todo table in the database that gets created by Django whenever we run the command `python manage.py migrate`.
In the `models.py` in the `todo` app, we will create our `Todo` model will the below code:

```
from django.db import models
# Create your models here.
class Todo(models.Model):
    title = models.CharField(max_length = 100)
    body = models.CharField(max_length = 100)
    is_completed = models.BooleanField(default=False)
    date_created = models.DateField(auto_created=True)
    last_modified = models.DateField(auto_now=True)
    def __str___(self):
        return self.title
```

To create an SQLite database and Todo table in the database, we will first run the command `./manage.py makemigrations` to create SQL queries that will be used to create a `Todo` table in the database. To create the Todo table in the database will run the command `./manage.py migrate`.
To make migrations run:

```
./manage.py makemigrations
```

To apply the migrations run:

```
./manage.py migrate
```

To get a better understanding of the Django migrations, read the documentation here.

Serializing the Todo model

To convert the Python objects obtained from the database to the JSON format needed by our endpoints and back to the Python objects that can be mapped to our database tables, we will subclass the Django rest framework `Serializer.ModelSerializer` class for easier conversion.

In the `todo` app directory, create a file `serializers.py`, where we will write our `TodoSerializer` code.

```
from rest_framework import serializers
from todo.models import Todo
class TodoSerializer(serializers.ModelSerializer):
    class Meta:
        model = Todo
        fields = "__all__"
```

We only need to pass in the model that we need to serialize and the fields to serialize as meta fields. In our case, we are serializing all of the fields from the `Todo` model that's why we are passing `__all__` to our `fields` variable.

Creating the TodoAPIView

In the `views.py` file in our `todo` app directory, we are going to write the logic for the CRUD functionality for our app. Django Rest framework comes with inbuilt classes that make building the CRUD functionality very easy.

We start by importing the Django rest framework classes that we'll use to create our CRUD API. We just need to create a class for each of our crud endpoints and add in the `queryset` and the `serializer_class` for our `Todo` model.

```
from django.shortcuts import render
from rest_framework.generics import ListAPIView
from rest_framework.generics import CreateAPIView
from rest_framework.generics import DestroyAPIView
from rest_framework.generics import UpdateAPIView
from todo.serializers import TodoSerializer
from todo.models import Todo
# Create your views here.
class ListTodoAPIView(ListAPIView):
    """This endpoint list all of the available todos from the database"""
    queryset = Todo.objects.all()
    serializer_class = TodoSerializer
class CreateTodoAPIView(CreateAPIView):
    """This endpoint allows for creation of a todo"""
    queryset = Todo.objects.all()
    serializer_class = TodoSerializer
class UpdateTodoAPIView(UpdateAPIView):
    """This endpoint allows for updating a specific todo by passing in the id of
the todo to update"""
    queryset = Todo.objects.all()
    serializer_class = TodoSerializer
class DeleteTodoAPIView(DestroyAPIView):
    """This endpoint allows for deletion of a specific Todo from the database"""
    queryset = Todo.objects.all()
    serializer_class = TodoSerializer
```

Creating URL paths for our Todo endpoints

The URLs allows us to interact with the various `Todo` crud views. In the `todo` app directory create a file `urls.py` where we will write URLs for various endpoints.

```
from django.urls import path
from todo import views
urlpatterns = [
    path("",views.ListTodoAPIView.as_view(),name="todo_list"),
    path("create/", views.CreateTodoAPIView.as_view(),name="todo_create"),
    path("update/<int:pk>/",views.UpdateTodoAPIView.as_view(),name="update_todo"),
    path("delete/<int:pk>/",views.DeleteTodoAPIView.as_view(),name="delete_todo")
]
```

In the `urls.py` in the `django_todo` project directory, let's added the base URL for our app so that the `django_todo` project can be aware of the `todo` app URLs.

```
from django.contrib import admin
from django.urls import path
from django.urls import include
urlpatterns = [
    path('admin/', admin.site.urls),
    path('api/v1/todo/',include("todo.urls"))
]
```

Testing the endpoints

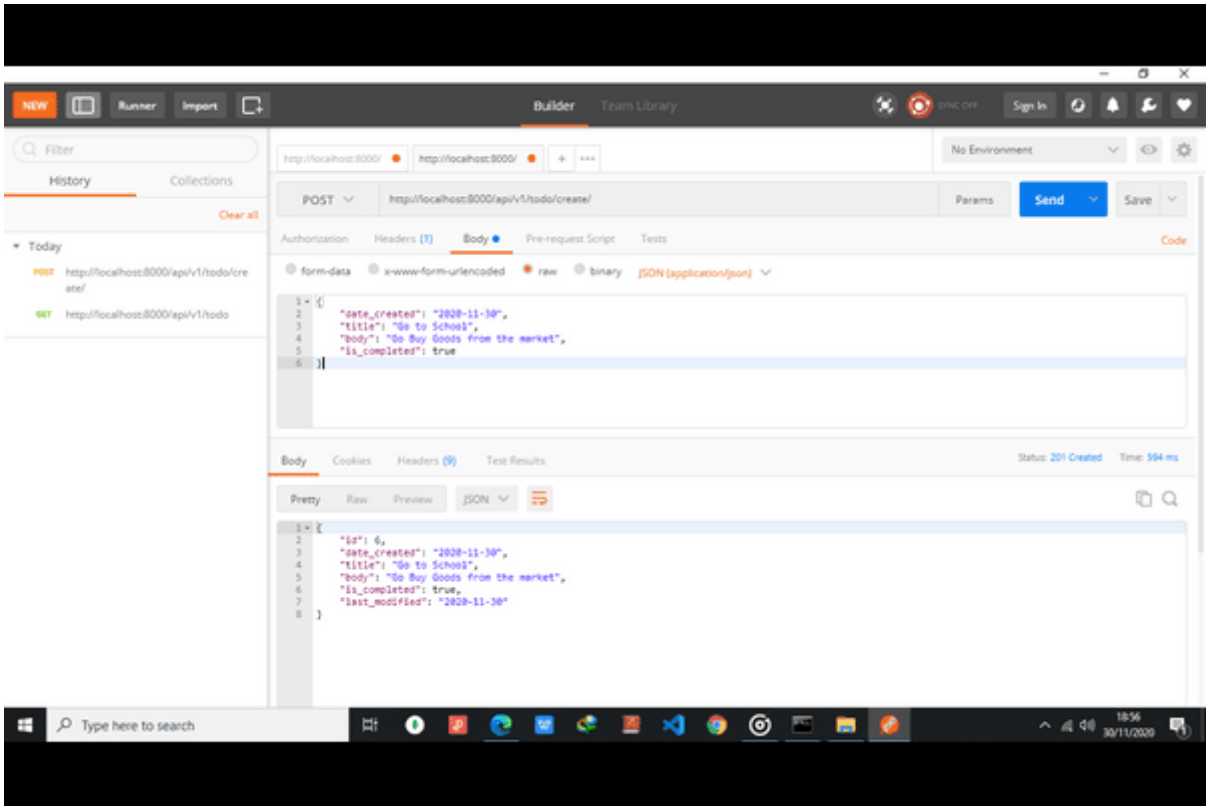
Before testing our API endpoints let's make sure our development server is up and running. To start the development server run the command `./manage.py runserver` in the root folder of our project where `manage.py` file exists.

```
$ ./manage.py runserver
```

To create a new Todo we make a POST request to `http://localhost:8000/api/v1/todo/create/` with the new Todo object.

```
{
    "date_created": "2020-11-19",
    "title": "Go to School",
    "body": "Go Buy Goods from the market",
    "is_completed": true
}
```

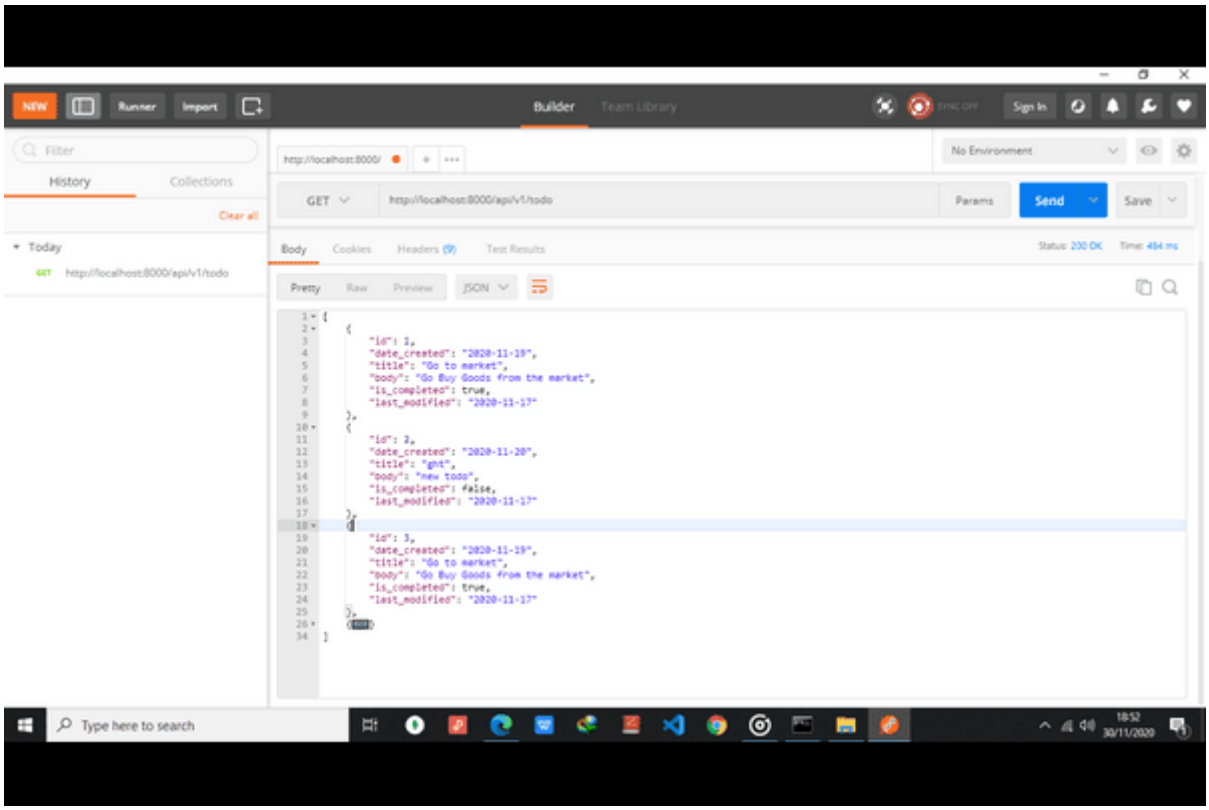
Sample postman POST request:



Making a GET request to `http://localhost:8000/api/v1/todo` in postman returns a list of `todos`.

```
[
    {
        "id": 1,
        "date_created": "2020-11-19",
        "title": "Go to market",
        "body": "Go Buy Goods from the market",
        "is_completed": true,
        "last_modified": "2020-11-17"
    }
]
```

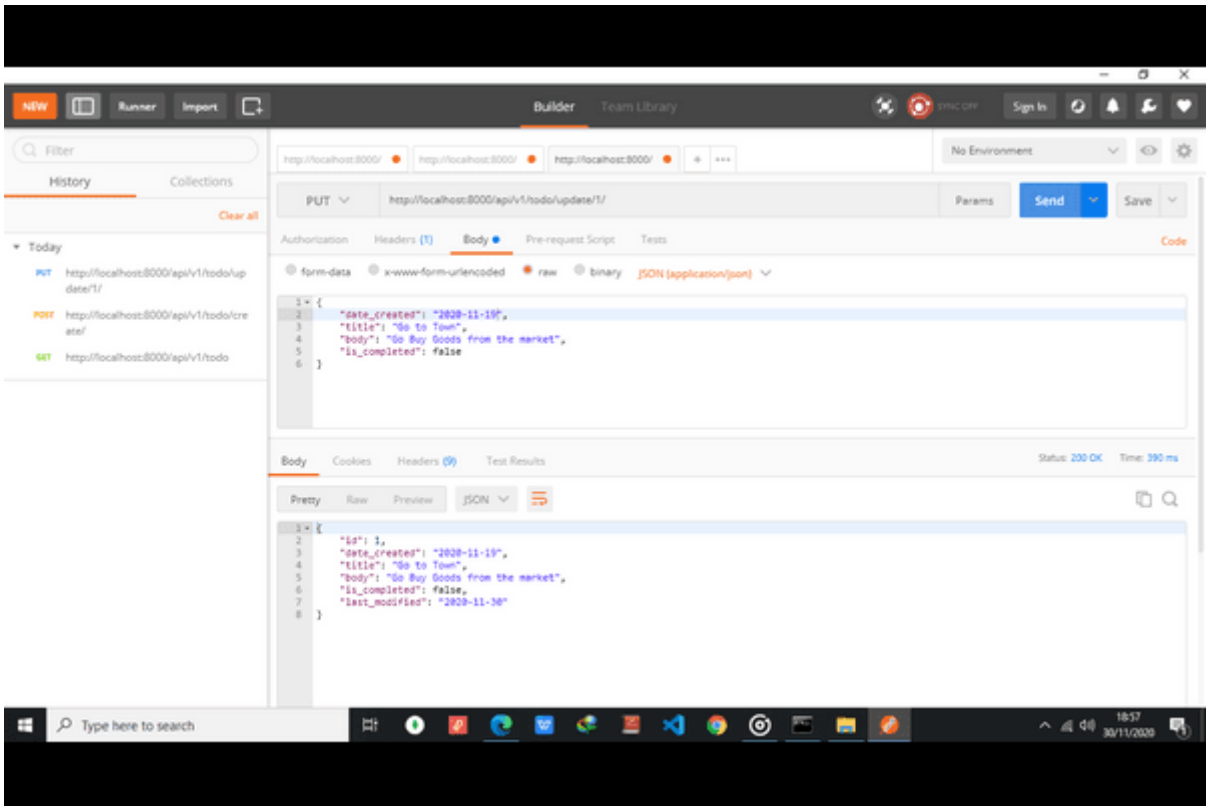
Sample postman GET request:



To Update a Todo we make a PUT request to `http://localhost:8000/api/v1/todo/update/1/` with the Todo object fields to update and pass in Todo ID as a URL parameter.

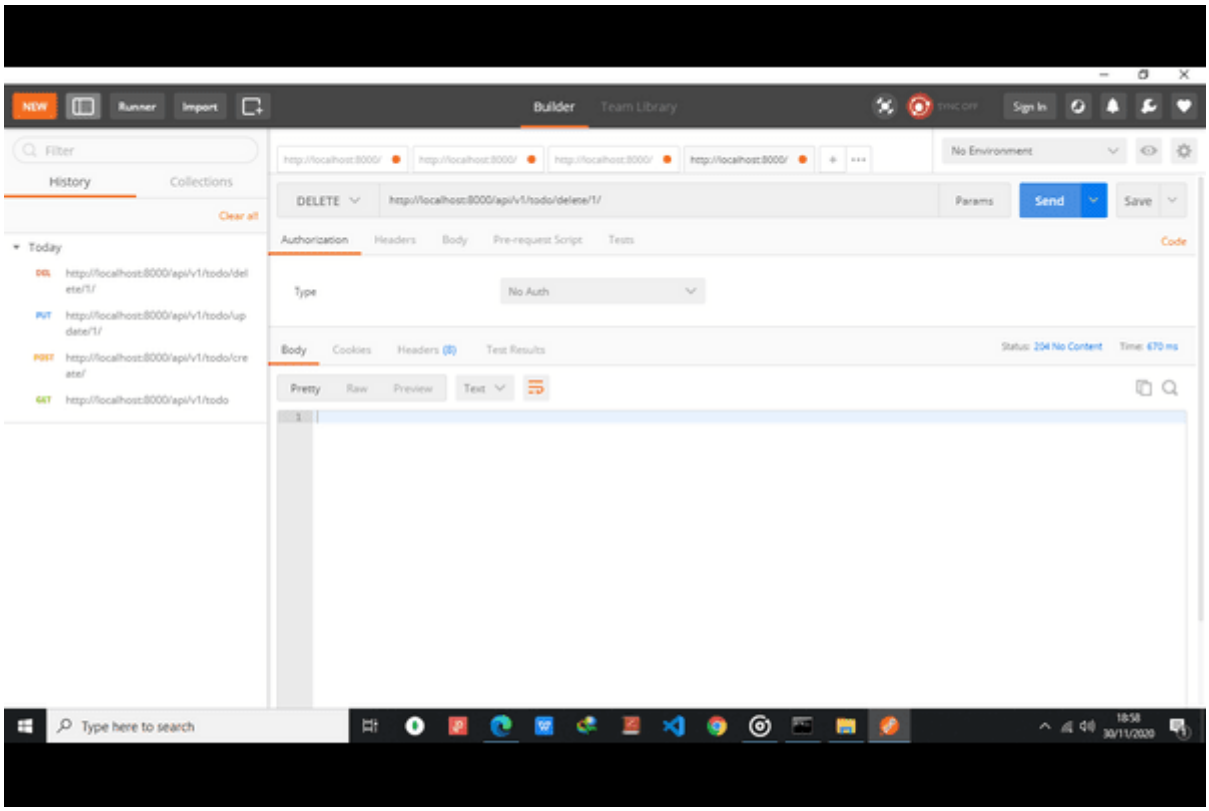
```
{
    "date_created": "2020-11-19",
    "title": "Go to Town",
    "body": "Go Buy Goods from the market",
    "is_completed": false
}
```

Sample postman PUT request:



To Delete a Todo we make a DELETE request to `http://localhost:8000/api/v1/todo/delete/1/` passing the ID of the Todo to delete as URL parameter.

Sample postman DELETE request:



Documenting Todo endpoints

It's a good practice to provide documentation for the various endpoints that we create, this makes it easier for other people to use our API endpoints.

We will use coreapi to document our endpoints. To install coreapi and plug it into our app we run the command `pip3 install coreapi` in the terminal.

On the `setting.py` file in `django_todo` project directory add `coreapi` to the installed apps list and the add the below rest framework configuration to enable documentation autogeneration.

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'todo',
    'rest_framework',
    'coreapi',
]

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

Finally we create a URL to our documentation page by adding a URL configuration in the `urls.py` file in the `django_todo` project directory.

```
from django.contrib import admin
from django.urls import path
from django.urls import include
from rest_framework.documentation import include_docs_urls
urlpatterns = [
    path('admin/', admin.site.urls),
    path('api/v1/todo/',include("todo.urls")),
    path('docs/', include_docs_urls(title='Todo Api')),
]
```

By visiting `http://127.0.0.1:8000/docs/` in the browser we'll be able to see the full documentation of our Todo CRUD API endpoints.

Conclusion

We now understand how to create and document our restful endpoint APIS in Django. Go ahead and clone the repos django_todo to view the full source code of the project and add new fields to our Todo model.

In our next article, we will secure our endpoints and add social authentication to our app.