Technical guide

Create a RESTful API With Django REST Framework

If you have used a digital device to send and receive information, you used an API. Developers create APIs to enable users to interact with data from their applications.

Creating a RESTful API (REST API) is a convenient way to share information. REST APIs have defined standards regulating data sharing between devices. To understand how REST APIs work, let's build one from scratch.

In this guide, you will use the Django REST Framework to build a REST API and use it to display data from a database. You will build a basic structure to handle GET and POST requests and data authentication.

What Is a REST API

To benefit from this guide, read about what is a REST API and the advantages of using it compared to other types of APIs.
You should also have the following pre-installed:

- Install the latest version of Python
- Install the latest version of pip
- Install Pipenv (but you can use venv)
- Install the latest version of Django
- Install a code editor. (I'm using VScode)

Now that you have installed all the essential software, let's get started.

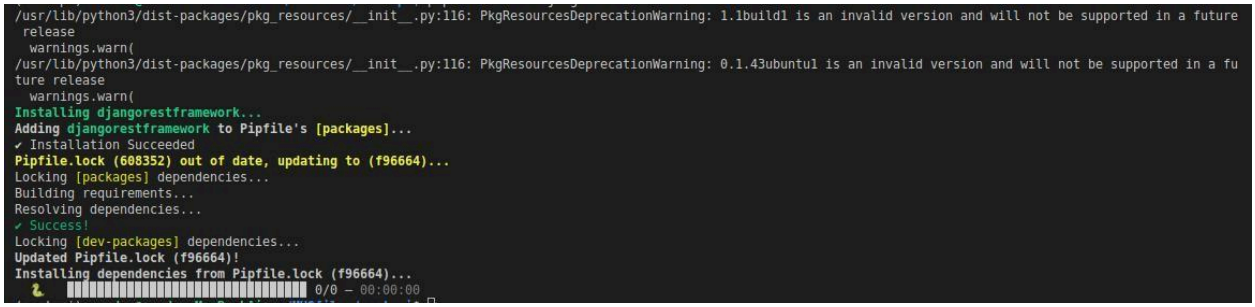
1. Install Django REST Framework

Django REST Framework is a powerful toolkit used to build and configure web APIs. The framework's customizable features make it popular among developers building REST APIs.

Install Django REST Framework with the following command:

```
pipenv install djangorestframework
```

A successful installation will display an installation success message as illustrated in the following image:



Next, start building the REST API by first creating a Django app.

2. Create a Django App

Create a food application that collects names and descriptions of popular Kenyan foods. Then, create an API to fetch requests from a database to enable users to interact with our data.

Django apps come equipped with an SQLite database, so you do not have to install another database.

To create a Django app, first create a project called **food** with the following command:

```
django-admin startproject food
```

Next, create a Django app called **kenyanfood**

```
django-admin startapp kenyanfood
```

3. Register the App Project Settings

Register the **kenyanfood** app in the project settings under the **INSTALLED APPS** array. If you skip this step, Django will not recognize the app.

Also, register the Django REST framework in the same settings as shown in the following image:

```
30
31  # Application definition
32
33  INSTALLED_APPS = [
34      'django.contrib.admin',
35      'django.contrib.auth',
36      'django.contrib.contenttypes',
37      'django.contrib.sessions',
38      'django.contrib.messages',
39      'django.contrib.staticfiles',
40      'kenyanfood',
41      'rest_framework',
42  ]
43
```

4. Register App URLs

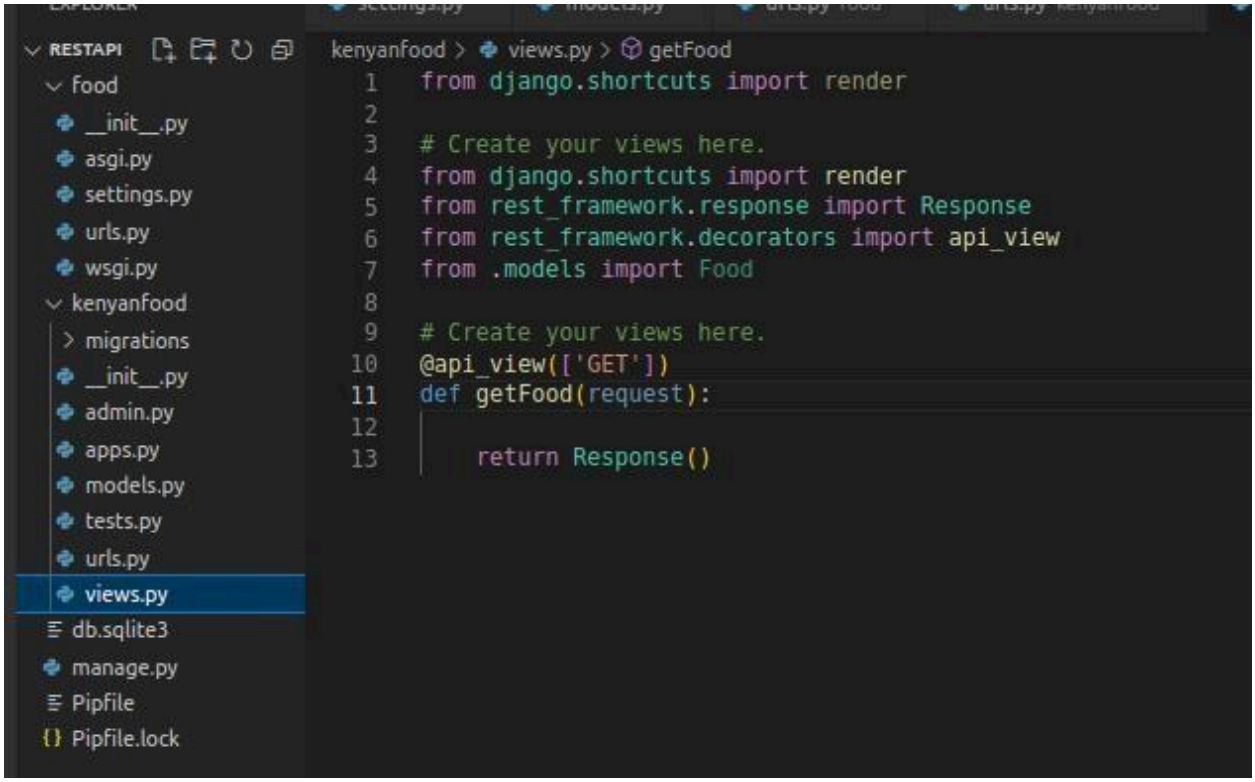
Register **Kenyanfood** app URLs in the project **urls.py** file as illustrated below:

```
7       1. Add an import:  from my_app import views
8       2. Add a URL to urlpatterns:  path('', views.home, name='home')
9   Class-based views
10      1. Add an import:  from other_app.views import Home
11      2. Add a URL to urlpatterns:  path('', Home.as_view(), name='home')
12  Including another URLconf
13      1. Import the include() function: from django.urls import include, path
14      2. Add a URL to urlpatterns:  path('blog/', include('blog.urls'))
15  """
16  from django.contrib import admin
17  from django.urls import path,include
18
19  urlpatterns = [
20      path('admin/', admin.site.urls),
21      path('', include('kenyanfood.urls')),
22  ]
23
```

5. Create a View for the API

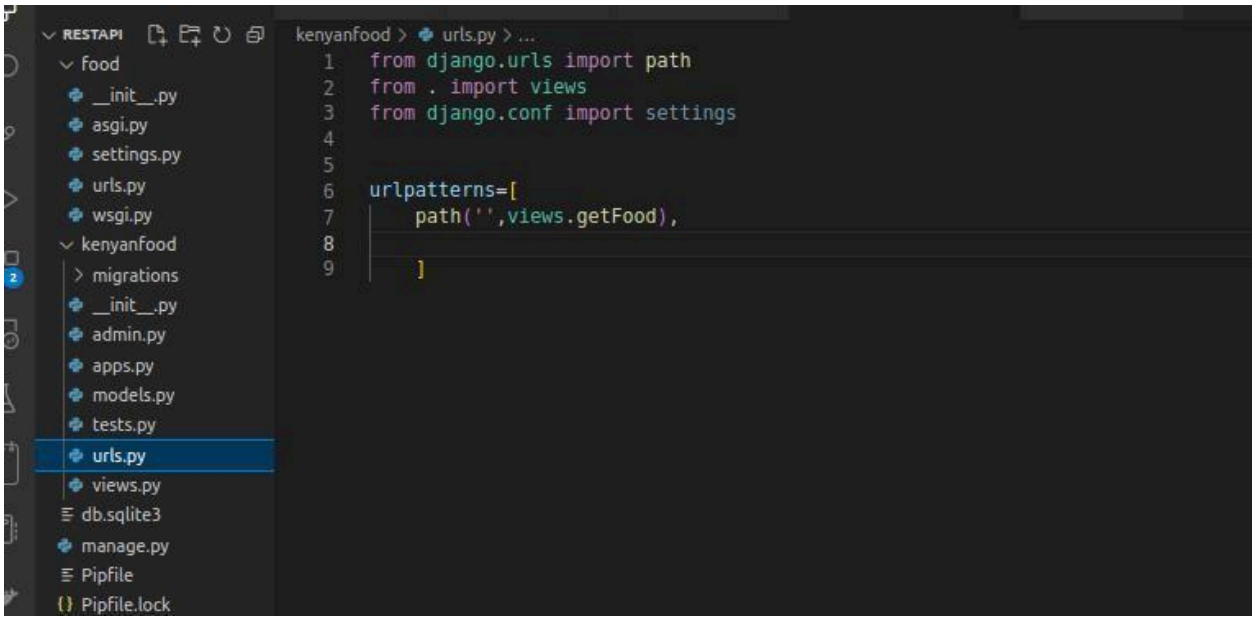
Create a dummy view in the App **views.py** file, so the app does not throw errors. First, import the **Response** object and **@apiview** decorator from the Django REST framework.

Response helps return sterilized data in **JSON** format while the **@apiview** displays the API.



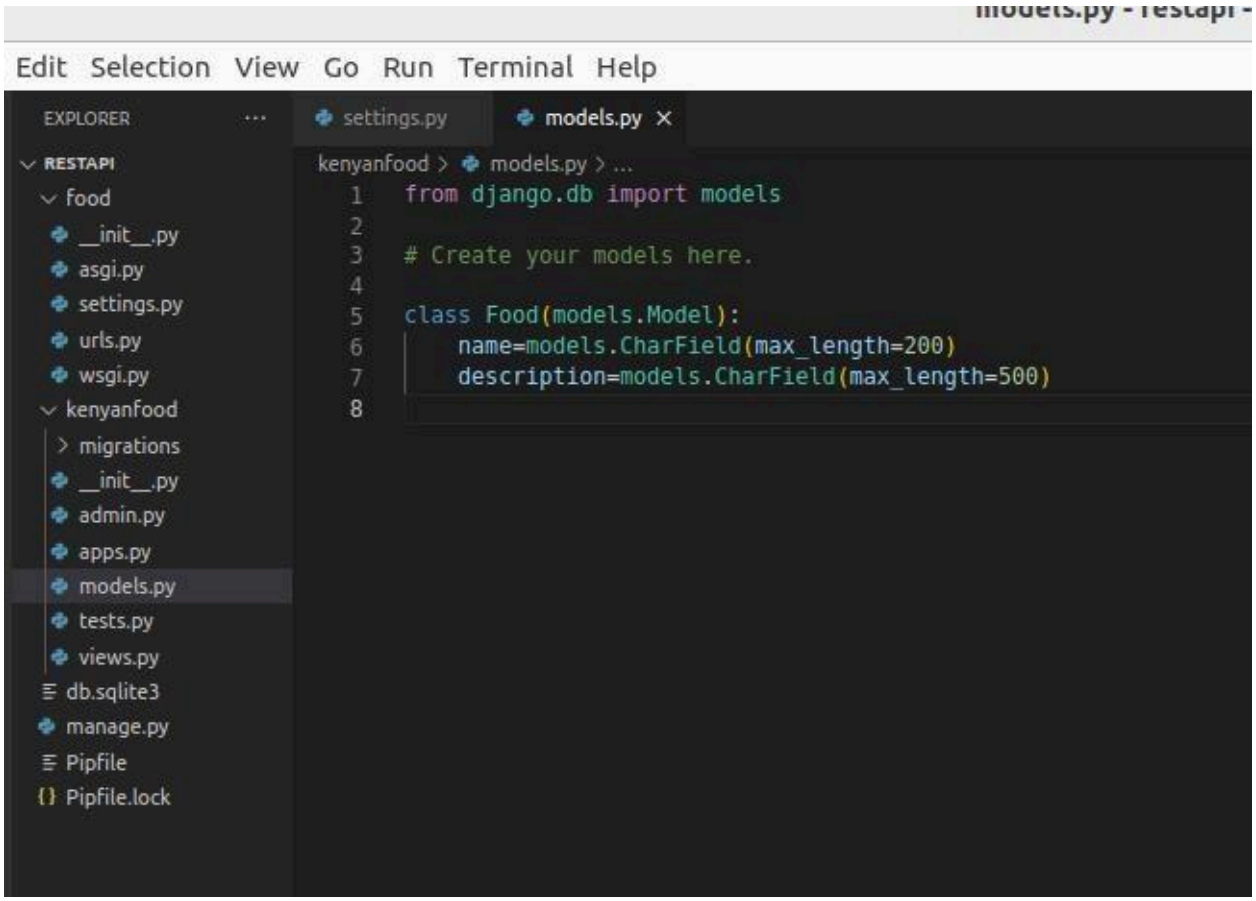
6. Create a URL Path for the App

Create a URL path for the API view. The `getFood` endpoint displays the **kenyanfood** data.

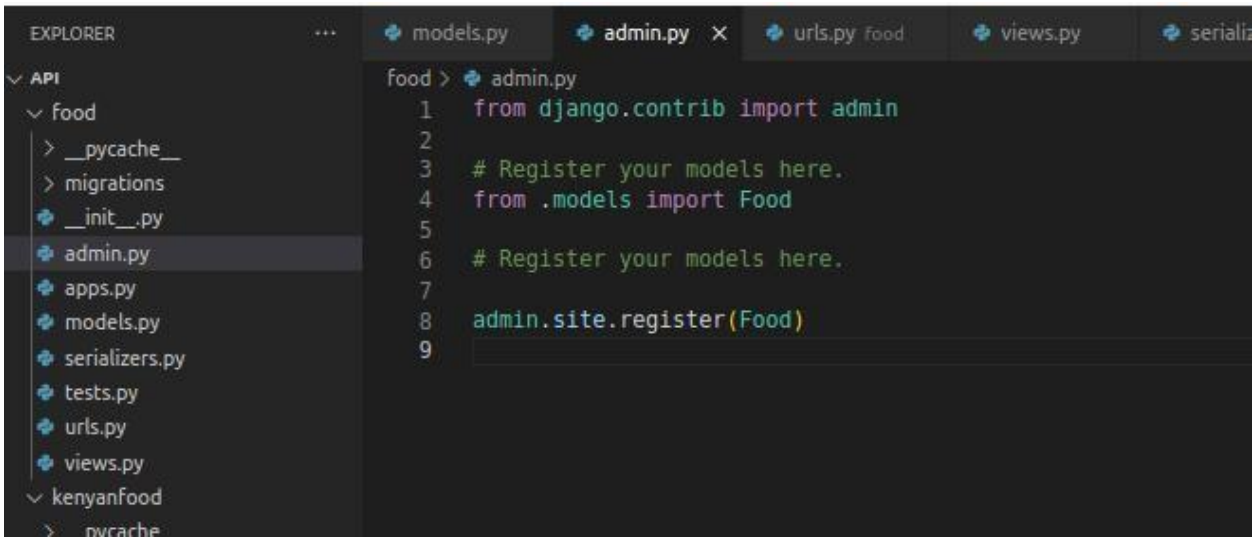


7. Create a Model for the App

The App model class is called **Food.** It would look like the following image:



Register the model in the app **admin.py** file as shown in the following image:



8. Make Migrations

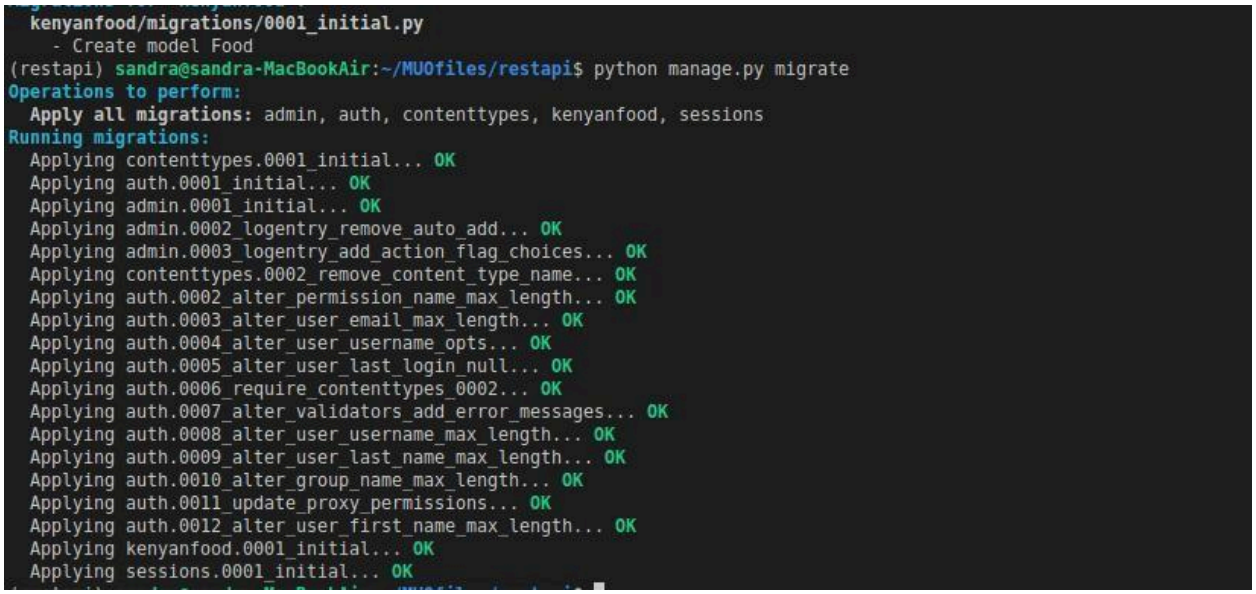
Next, **migrate** the app to create tables in the **SQLite** database.
Migrate using the following command:

```
Python manage.py makemigrations kenyanfood
```

Next, apply these migrations by running the following command:

```
Python manage.py migrate
```

A successful migration will look like this:



Successful migrations mean that the database has created tables for the **kenyanfood** App. Next, enter data into the **kenyanfood** database table.

9. Add Data to the Database

Use the **Django admin** Graphical User Interface(GUI) to enter data into the database. **Django admin** has a great interface to visualize and manage your application's data.

You can also add data to the database using the Python shell on the command line. In this guide, use the **Django admin** interface.

Use the following command to set up the **Django admin**

```
python manage.py createsuperuser
```

When prompted, enter your **username, email, and password**. You can then open the admin page using the following link http://127.0.0.1:8000/admin/

You will see the login page.

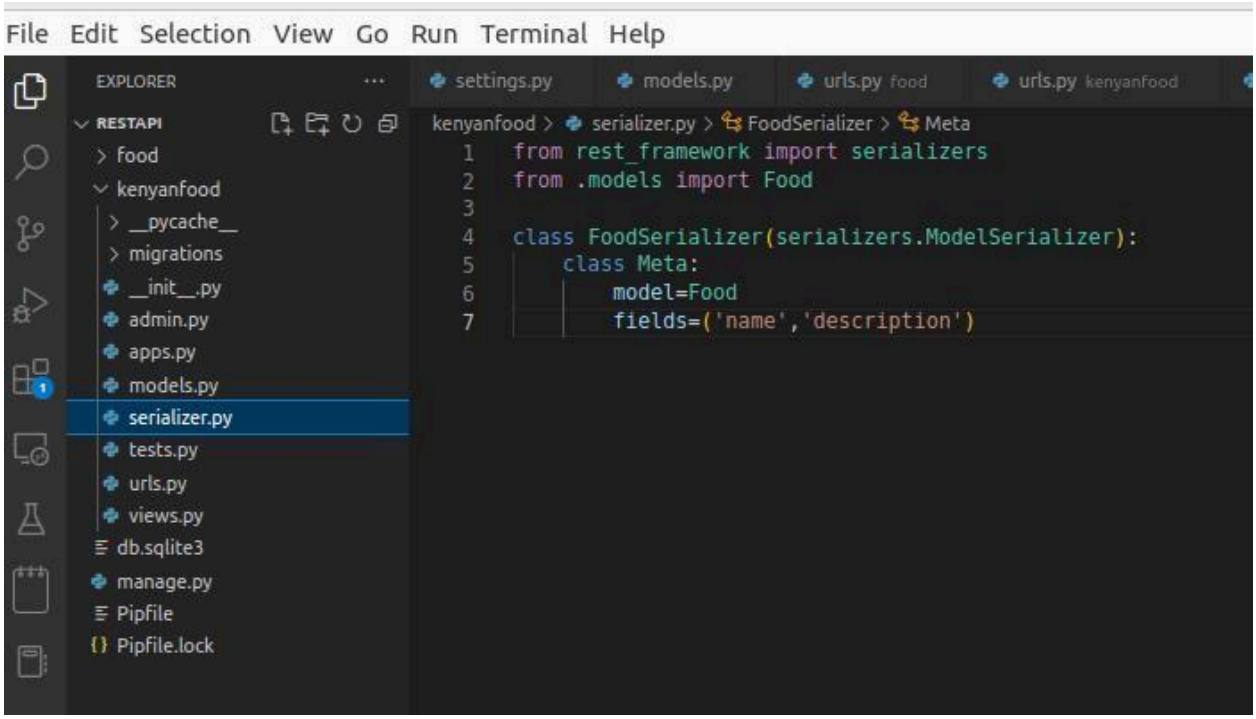
Once you log in, navigate to the Django administration interface with **Groups** and **Users** model. The two are for authentication. The **Food** model is in the section below it.

You can add and delete **Food** items from the database from the admin page. Add a few Kenyan delicacies, such as Ugali, Pilau, and Chai, to the database.
Now that you have populated the database, create the API.

10. Serialize the Model

Serializers convert complex Django models to **JSON** objects, making data easily read on the API. Serializing makes data more readable on the API.

Create a new file in the app called **serializer.py**



Here, you import the **serializers** module from the **rest_framework** package and create a **FoodSerializer** class that inherits from the **ModelSerializer** class.

Next, specify the **Food** model you want to serialize and the fields to add to the API.

11. Update the View

Next, update the API view with the **serializer** and **Food** models.

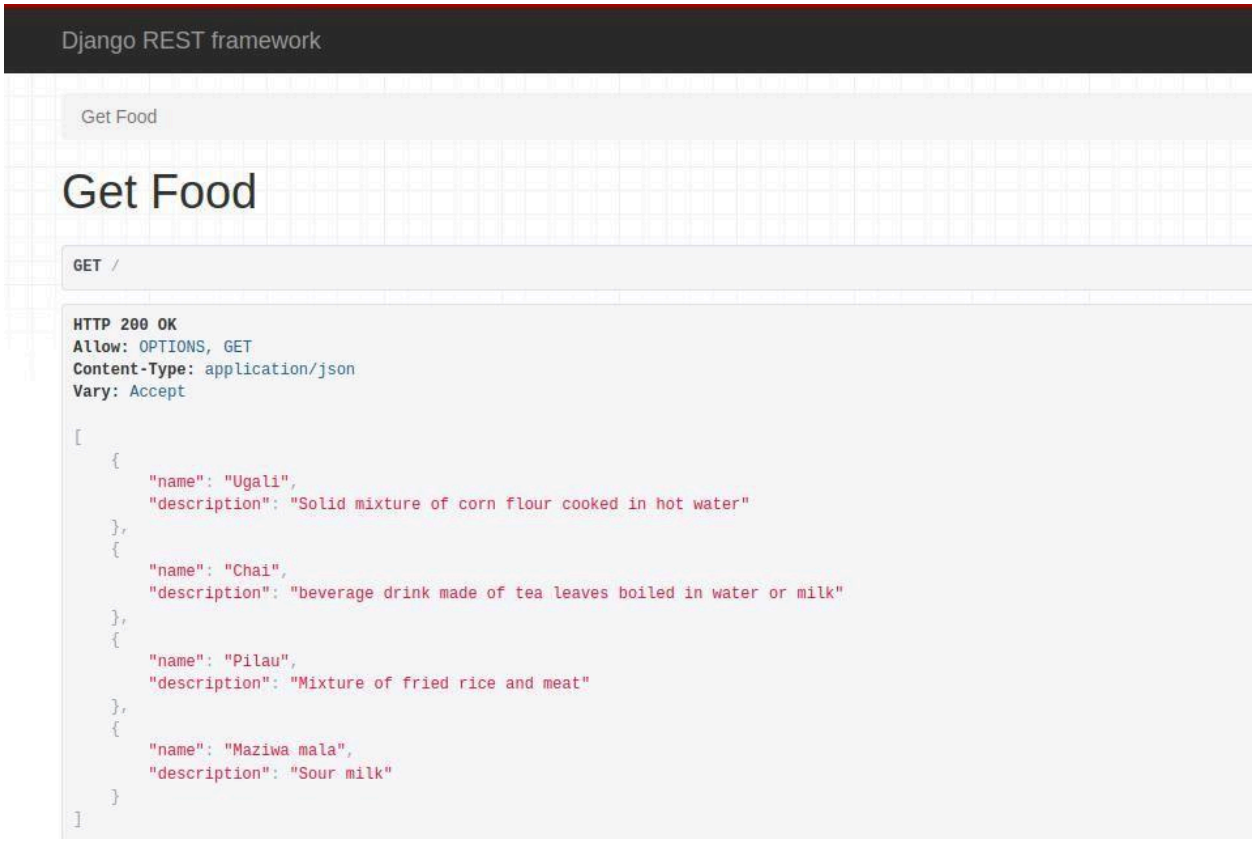
First, define a **GET** method to retrieve all our data from the database with **Food.Objects.all()** function. Then, the data is serialized and returned as a response in **JSON** format.

settings.py | models.py | urls.py food | urls.py kenyanfood | admin.py | serializer.py

kenyanfood > views.py > ...

```python
from django.shortcuts import render
from rest_framework.response import Response
from rest_framework.decorators import api_view
from .models import Food
from .serializer import FoodSerializer

# Create your views here.
@api_view(['GET'])
def getFood(request):
    food=Food.objects.all()
    serializer=FoodSerializer(food,many=True)
    return Response(serializer.data)
```

When you navigate to the server URL link http://127.0.0.1:8000/ , you will see the API displaying data from the database.



Congratulations, you have created a REST API!

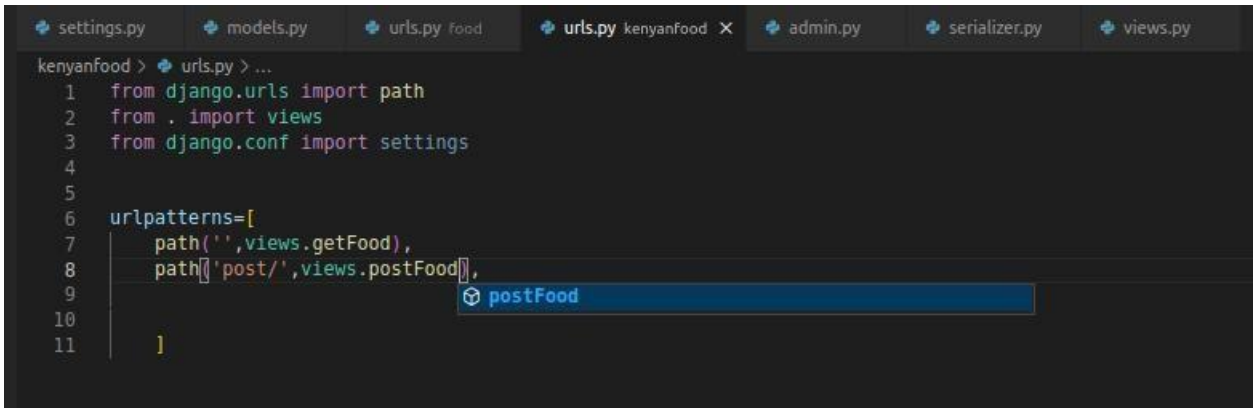
12. Add Data with POST Method

Test whether the REST API can add data to the database.

First, define a **POST** method in our view.

```python
from rest_framework.response import Response
from rest_framework.decorators import api_view
from .models import Food
from .serializer import FoodSerializer

# Create your views here.
@api_view(['GET'])
def getFood(request):
    food=Food.objects.all()
    serializer=FoodSerializer(food,many=True)
    return Response(serializer.data)

@api_view(['POST'])
def postFood(request):
    serializer=FoodSerializer(data=request.data)
    if serializer.is_valid():
        serializer.save()
    return Response(serializer.data)

```

Then, add a path in the app **urls.py** to create an endpoint for the API **POST** functionality.



Next, navigate to **the** URL http://127.0.0.1:8000/post to find the **POST** endpoint. Add data to the database in **JSON** format in the **Content** section and click the **POST** button**.**

Add one more food item **{"name":"Maziwa mala","description":"Sour milk"}**

The data will display in **JSON** format as shown in the following image.

```
POST /post/
```

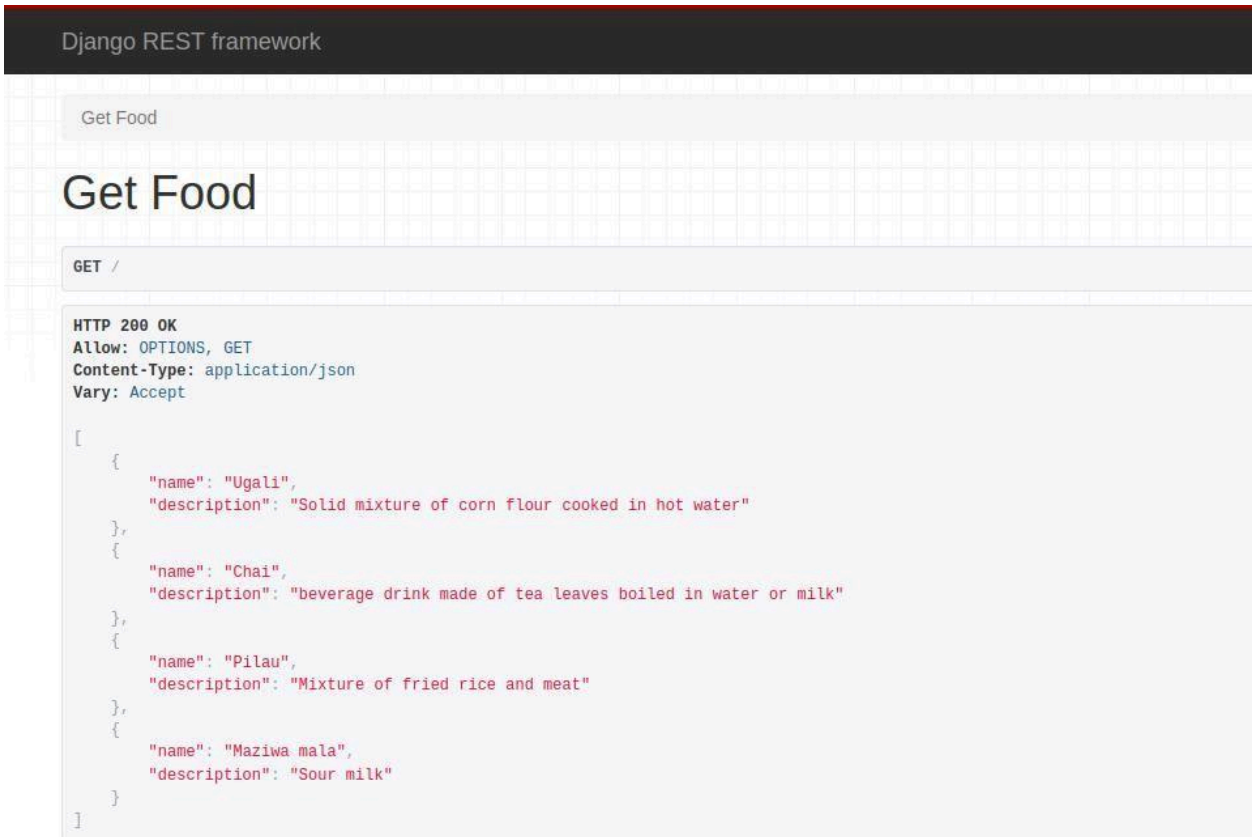
```
HTTP 200 OK
Allow: OPTIONS, POST
Content-Type: application/json
Vary: Accept

{
    "name": "Maziwa mala",
    "description": "Sour milk"
}
```

Media type: application/json

Content:

POST

Now, if you navigate back to the **GET** endpoint, you will see the food‘**Maziwa mala’** and Its description added.



You now have a REST API that can display and add items to our application. How about experimenting with other **CRUD** methods? Working with **UPDATE** and **DELETE** methods will increase the functionality of your REST API.

Learn what more you can do with Django REST Framework on the official open-source project page.

You can also learn how to consume REST APIs to add features to your app.

How To Create a REST API With Django

You can now create a REST API using the Django REST Framework. First, create an App with a model, serialize the data, and create a view function. Next, include URL endpoints to visualize the data in JSON format. Building REST APIs with Django is a convenient way to share data and give your users a great customer experience.