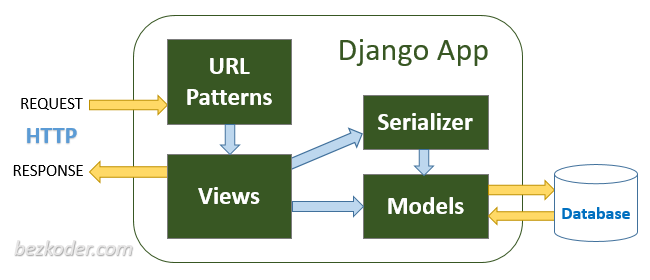
**Create a REST API With Django REST Framework**

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

Creating a 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, you can build one from scratch.

**Django Rest Api Architecture**

Let’s look at the architecture of our Django Rest Apis App:



* HTTP requests will be matched by **Url Patterns** and passed to the **Views**
* **Views** processes the HTTP requests and returns HTTP responses (with the help of **Serializer**)
* **Serializer** serializes/deserializes data model objects
* **Models** contains essential fields and behaviors for CRUD Operations with Database

## Using Django With a REST API

You can [**use a REST API to fetch structured data**](https://www.makeuseof.com/what-is-rest-api/) over HTTP. Like many languages and frameworks, Django lets you build your own API and consume others.

You should also have the following pre-installed:

1. The latest version of python.
2. [**The latest version of pip**](https://www.makeuseof.com/tag/install-pip-for-python/).
3. Pipenv (although you can use venv instead if you want.)
4. The latest version of Django.

Once you’ve installed all the essential software, you’re ready to begin.

## 1.Create virtual Environment and Install Django and Django REST Framework

[**Django REST framework**](https://www.django-rest-framework.org/) is a powerful toolkit you can use to build and configure web APIs. Its customizable features make it a popular choice to build REST APIs.

**Step 1: pip install virtualenv**

First, You need to install the virtualenv, Open a command prompt by going to the project folder directory and typing CMD. After opening the CMD type ” **pip install virtualenv** “.

**Step 2: virtualenv env**

Then, after installing virtualenv just type “**virtualenv env**” and enter.

**Step 3: cd env/Scripts**

Next, just type ” **cd env/Scripts**” and press enter.

**Step 4: activate**

Next, you need to type “**activate**” then press enter.

**Step 5: cd ../..**

Next, just type ” **cd ../..** ” and press enter.

**Step 6: Install Django**

Next, type the command “**pip install django**”.

**Step 7: Install the Django REST framework** with the following command:

pipenv **install** djangorestframework

## 2. Create a Django Project

The following instructions will explain how to create a food application to gather names and descriptions of popular Kenyan foods. The API will fetch requests from a database to enable users to interact with that data.

Django apps come equipped with an SQLitedatabase, so you don't 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:

# Application definition  
   
INSTALLED\_APPS = [  
    'django.contrib.admin',  
    'django.contrib.auth',  
    'django.contrib.contenttypes',  
    'django.contrib.sessions',  
    'django.contrib.messages',  
    'django.contrib.staticfiles',  
    'kenyanfood',  
    'rest\_framework',  
]

## 4. Register App URLs

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

**from** django.contrib **import** admin  
**from** django.urls **import** path, include  
   
urlpatterns = [  
    path('admin/', admin.site.urls),  
    path('', **include**('kenyanfood.urls')),  
]

## 5. Create a View for the API

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

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

**from** django.shortcuts **import** render  
**from** rest\_framework.response **import** Response  
**from** rest\_framework.decorators **import** api\_view  
   
# Create your views here.  
@api\_view(['GET'])  
**def** **getFood**(request):  
    **return** Response()

## 6. Create a URL Path for the App

Create a URL path for the API view you created. This endpoint displays the **kenyanfood**data.

**from** django.urls **import** path  
**from** . **import** views  
**from** django.conf **import** settings  
   
urlpatterns = [  
    path('', views.getFood),  
    ]

## 7. Create a Model for the App

The app’s model class is called **Food.** It should look like this:

**from** django.db **import** models  
   
# Create your models here.  
**class** **Food**(models.Model):  
    name = models.CharField(max\_length=200)  
    description = models.CharField(max\_length=500)

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

**from** django.contrib **import** admin  
**from** .models **import** Food  
   
# Register your models here.  
**admin**.site.register(**Food**)

## 8. Make Migrations

Next, **migrate** the app to create tables in the **SQLite** database. You can do this using the following command:

**python** **manage**.py **makemigrations** **kenyanfood**

Next, apply these migrations by running this 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.

## 9. Add Data to the Database

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

Alternatively, you can use the python shell on the command line to enter data manually into the database. In this guide, you will use the Django admin interface.

Use the following command to set up Django admin:

**python** **manage**.py **createsuperuser**

When prompted, enter your **username, email, and password**. You can then open the admin page using the link below:

http://127.0.0.1:8000/admin/

You will see the login page:

Once you log in, you will see the Django administration interface with **Groups**and **Users** model. These are both for authentication; the **Food** model is in the section below.

You can add and delete **Food** items from the database from the admin page. Add some Kenyan delicacies, such as Ugali, Pilau, and Chai, to the database.

Now that the database has data, 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**

**from** rest\_framework **import** serializers  
**from** .models **import** Food  
   
**class** **FoodSerializer**(serializers.ModelSerializer):  
    **class** **Meta**:  
        model=Food  
        fields=('name','description')

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 you want 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 the data from the database with **Food.Objects.all()** function. Then serialize the dataand returned it as a response in **JSON**format.

**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)

Then, navigate to the server URL link:

https://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 you can use the REST API to add data to the database.

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

@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.

urlpatterns = [  
    path('',views.getFood),  
    path('post/',views.postFood),  
]

Next, navigate tothis URL:

https://127.0.0.1:8000/post

You will see the **POST**endpoint. Add data to the database in **JSON**format in the **Content** section and click the **POST** button. For example, add a new food item with this structure:

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

You will see the data displayed in red in **JSON**format.

Now, if you navigate back to the **GET**endpoint[**http://127.0.0.1:8000/**](http://127.0.0.1:8000/), you will see the food ‘**Maziwa mala,’** and Its description added.