Create a folder.

Inside the folder, create two folders – one for frontend and one for backend.

Inside the frontend folder, you can create react project

Inside the backend folder, you can create django project

**Steps for Django Backend Side:**

**pip install django django-cors-headers**

**django-cors-headers** is a Django application used for handling Cross-Origin Resource Sharing (CORS) headers in Django projects. CORS is a security feature implemented by web browsers to restrict scripts running on a web page from making requests to another domain. This restriction is known as the same-origin policy.

However, in certain cases, you might want your web application to make requests to resources hosted on a different domain. For example, if your frontend code is hosted on a different server or domain from your Django backend, you'll need to configure CORS to allow communication between them.

INSTALLED\_APPS = [

    ...

    'corsheaders',

'myapp',

    ...

]

MIDDLEWARE = [

    ...

    'corsheaders.middleware.CorsMiddleware',

    ...

]

CORS\_ORIGIN\_ALLOW\_ALL = True  # This allows requests from any origin

# Or, you can specify specific origins:

CORS\_ALLOWED\_ORIGINS = [

    "http://localhost:3000", # React development server

]

# You can configure other CORS settings as well, such as allowed methods, headers, etc.

Then you can define models.py

# myapp/models.py

from django.db import models

class Item(models.Model):

    name = models.CharField(max\_length=255)

    description = models.TextField()

    is\_available = models.BooleanField(default=False)

    price=models.DecimalField(max\_digits=10, decimal\_places=2)

    def \_\_str\_\_(self):

        return self.name

create database table using django:

python manage.py makemigrations

python manage.py migrate

Views.py

# myapp/views.py

from django.shortcuts import get\_object\_or\_404

from django.http import JsonResponse

from .models import Item

from django.forms import model\_to\_dict

def get\_items(request):

    items = Item.objects.all()

    data = {"items": list(items.values())}

    return JsonResponse(data)

def get\_item(request, item\_id):

    item = get\_object\_or\_404(Item, pk=item\_id)

    data = {"item": model\_to\_dict(item)}

    return JsonResponse(data)

This project has not used serializers.py to convert data into json format. Instead, it has imported JsonResponse in views.py to convert data from the database into json format.

We have used Function Based View

1. **get\_items**: This view retrieves all items from the database and returns them as JSON data. It first queries all items using **Item.objects.all()**, then converts the queryset to a list of dictionaries using **values()**, and finally wraps it in a JSON response using **JsonResponse**.
2. **get\_item**: This view retrieves a single item specified by its ID (**item\_id**). It uses Django's **get\_object\_or\_404** shortcut to retrieve the item from the database based on its primary key (**pk=item\_id**). If the item is not found, it returns a 404 HTTP response. Once the item is retrieved, it's converted to a dictionary using **model\_to\_dict**, and then wrapped in a JSON response using **JsonResponse**.

You can register models to admin.py

from django.contrib import admin

from .models import Item

# Register your models here.

admin.site.register(Item)

lis\_display = "\_\_all\_\_"

then we will be defining urls.py at application level.

urls.py/app

# myapp/urls.py

from django.urls import path

from .views import get\_items, get\_item

urlpatterns = [

    path('api/items/', get\_items, name='get\_items'),

    path('api/items/<int:item\_id>/', get\_item, name='get\_item'),

]

then we will be including app all urls to project level urls.py.

Urls.py/project

# myproject/urls.py

from django.contrib import admin

from django.urls import path, include

urlpatterns = [

    path('admin/', admin.site.urls),

    path('', include('myapp.urls')),

]

Create superuser for admin panel

Python manage.py createsuperuser

Now, you can add data in the admin panel

Now, run the django server with

python manage.py runserver

now, you can access data in the path [**http://127.0.0.1:8000/api/items/**](http://127.0.0.1:8000/api/items/)

**this will be a json data in json format. Not in API format.**

**To get API View, we need to install djangorestframework**

To create a browsable API for your function-based views (FBV) in Django, you can utilize Django REST Framework (DRF), which provides powerful tools for building Web APIs. Here's how you can integrate DRF with your existing views:

1. **Install Django REST Framework**: If you haven't already installed Django REST Framework, you can do so using pip:

pip install djangorestframework

1. **Update Django Settings**: Add **'rest\_framework'** to your **INSTALLED\_APPS** setting in **settings.py**:

INSTALLED\_APPS = [

    ...

    'rest\_framework',

]

1. **Create Serializers**: Serializers are used to convert complex data types, such as Django model instances, to native Python datatypes. In your case, you can define serializers for your **Item** model.

# myapp/serializers.py

from rest\_framework import serializers

from .models import Item

class ItemSerializer(serializers.ModelSerializer):

    class Meta:

        model = Item

        fields = '\_\_all\_\_'

1. **Update Views to Use Serializers**: Modify your existing views to use serializers to serialize the data:

# myapp/views.py

from rest\_framework.decorators import api\_view

from rest\_framework.response import Response

from .models import Item

from .serializers import ItemSerializer

@api\_view(['GET'])

def get\_items(request):

    items = Item.objects.all()

    serializer = ItemSerializer(items, many=True)

    return Response(serializer.data)

@api\_view(['GET'])

def get\_item(request, item\_id):

    item = get\_object\_or\_404(Item, pk=item\_id)

    serializer = ItemSerializer(item)

    return Response(serializer.data)

That’s it for Backend!!

Now, you can create frontend folder and create a react project.

Write component to fetch data from the server by using useEffect.

We just need to insert this url link of the Django like [**http://127.0.0.1:8000/api/items/**](http://127.0.0.1:8000/api/items/)

**Example:**

**App.js**

import React, { useState, useEffect } from 'react';

function App() {

  const [items, setItems] = useState([]);

  useEffect(() => {

    fetch('http://localhost:8000/api/items/')

      .then(response => response.json())

      .then(data => {

        console.log('Fetched data:', data);

        setItems(data.items);

      })

      .catch(error => console.error('Error fetching data:', error));

  }, []);

  return (

    <div style={styles.container}>

      <h1 style={styles.header}>Items</h1>

      <div style={styles.gridContainer}>

        {items.map(item => (

          <div key={item.id} style={styles.gridItem}>

            <strong>{item.name}</strong>

            <p>{item.description}</p>

            <p>Availability: {item.is\_available ? 'Yes' : 'No'}</p>

            <p>Price: Rs. {item.price}</p>

          </div>

        ))}

      </div>

    </div>

  );

}

export default App;