Python | Uploading images in Django

Last Updated : 03 Jun, 2024

In most websites, we often deal with media data such as images, files, etc. In Django, we can deal with the images with the help of the model field which is ImageField. In this article, we have created the app image_app in a sample

Prerequisite – Introduction to Django

Python

```
MEDIA_ROOT =  os.path.join(BASE_DIR, 'media')
  MEDIA_URL = '/media/'
```

MEDIA_ROOT is for server path to store files in the computer. **MEDIA_URL** is the reference URL for the browser to access the files over Http. In the urls.py we should edit the configuration like this

```
from django.conf import settings
from django.conf.urls.static import static
if settings.DEBUG:
        urlpatterns += static(settings.MEDIA_URL,
                              document_root=settings.MEDIA_ROOT)
```

A sample models.py should be like this, in that we have created a **Hotel model** which consists of hotel name and its image. In this project we are taking the hotel name and its image from the user for hotel booking website.

Python

```python
# models.py
 class Hotel(models.Model):
    name = models.CharField(max_length=50)
    hotel_Main_Img = models.ImageField(upload_to='images/')
```

Here upload_to will specify, to which directory the images should reside, by default django creates the directory under **media** directory which will be automatically created when we upload an image. No need of explicit creation of **media** directory. We have to create a forms.py file under image_app, here we are dealing with **model form** to make content easier to understand.

Python

```python
# forms.py
from django import forms
from .models import Hotel


class HotelForm(forms.ModelForm):

    class Meta:
        model = Hotel
        fields = ['name', 'hotel_Main_Img']
```

Django will implicitly handle the form verification's with out declaring explicitly in the script, and it will create the analogous form fields in the page according to model fields we specified in the models.py file. This is the advantage of model form. Now create a **templates** directory under image_app in that we have to create a html file for uploading the images. HTML file should look like this.

html

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
```

```
        <title>Hotel_image</title>
</head>
<body>
    <form method = "post" enctype="multipart/form-data">
        {% csrf_token %}
        {{ form.as_p }}
        <button type="submit">Upload</button>
    </form>
</body>
</html>
```

When making a POST request, we have to encode the data that forms the body of the request in some way. So, we have to specify the encoding format in the form tag. multipart/form-data is significantly more complicated but it allows entire files to be included in the data. The csrf_token is for protection against Cross-Site Request Forgeries. form.as_p simply wraps all the elements in HTML paragraph tags. The advantage is not having to write a loop in the template to explicitly add HTML to surround each title and field. In the views.py under image_app in that we have to write a view for taking requests from user and gives back some html page.

Python

```python
from django.http import HttpResponse
from django.shortcuts import render, redirect
from .forms import HotelForm

# Create your views here.


def hotel_image_view(request):

    if request.method == 'POST':
        form = HotelForm(request.POST, request.FILES)

        if form.is_valid():
            form.save()
            return redirect('success')
    else:
        form = HotelForm()
```

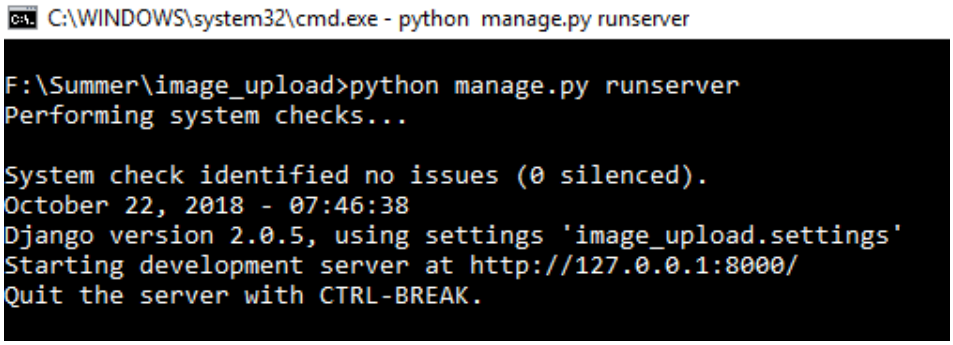
```
        return render(request, 'hotel_image_form.html', {'form':
    form})
19
20
21  def success(request):
22      return HttpResponse('successfully uploaded')
```

whenever the hotel_image_view hits and that request is POST, we are creating an instance of model form form = HotelForm(request.POST, request.FILES) image will be stored under request.FILES one. If it is valid save into the database and redirects to success url which indicates successful uploading of the image. If the method is not POST we are rendering with html template created. urls.py will look like this –

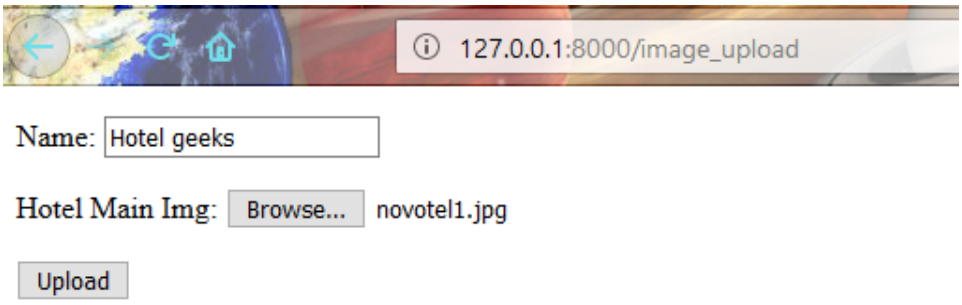
Python

```
1  from django.contrib import admin
2  from django.urls import path
3  from django.conf import settings
4  from django.conf.urls.static import static
5  from .views import hotel_image_view
6
7  urlpatterns = [
8      path('image_upload', hotel_image_view,
   name='image_upload'),
9      path('success', success, name='success'),
10 ]
11
12 if settings.DEBUG:
13     urlpatterns += static(settings.MEDIA_URL,
14
   document_root=settings.MEDIA_ROOT)
```

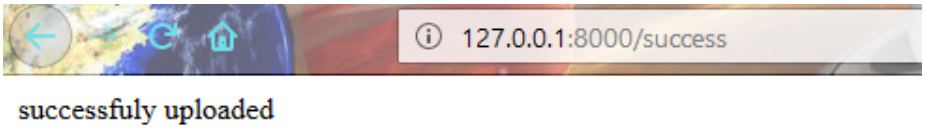
Output:



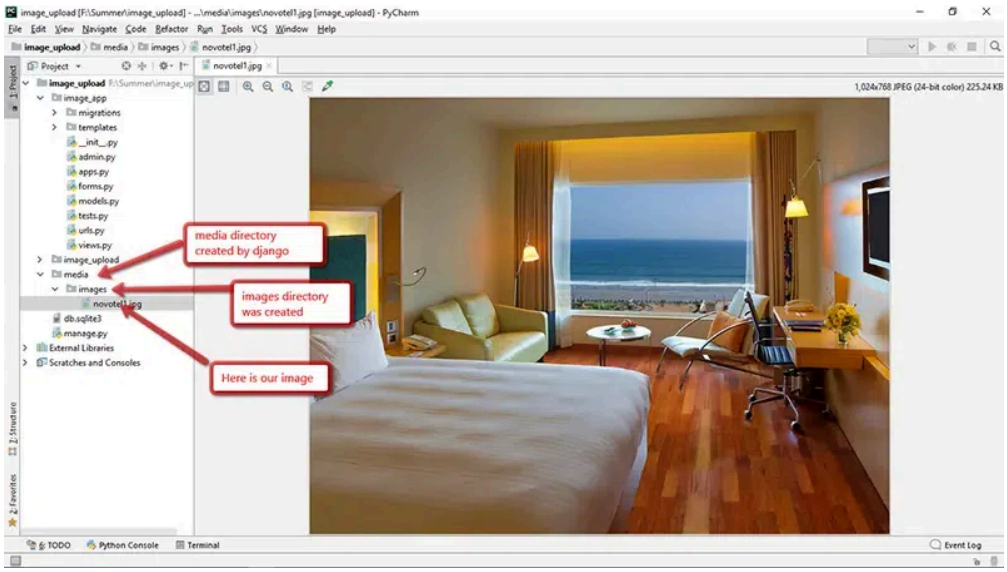
Now make the migrations and run the server. When we hit the URL in the browser, in this way it looks.



After uploading the image it will show success.



Now in the project directory media directory will be created, an images directory will be created and the image will be stored under it. Here is the final result.



Now we can write a view for accessing those images, for simplicity let's take example with one image and it is also applicable for many images.

Python

```python
# Python program to view
# for displaying images


def display_hotel_images(request):

    if request.method == 'GET':

        # getting all the objects of hotel.
        Hotels = Hotel.objects.all()
        return render((request, 'display_hotel_images.html',
                       {'hotel_images': Hotels}))
```

A sample html file template for displaying images.

html

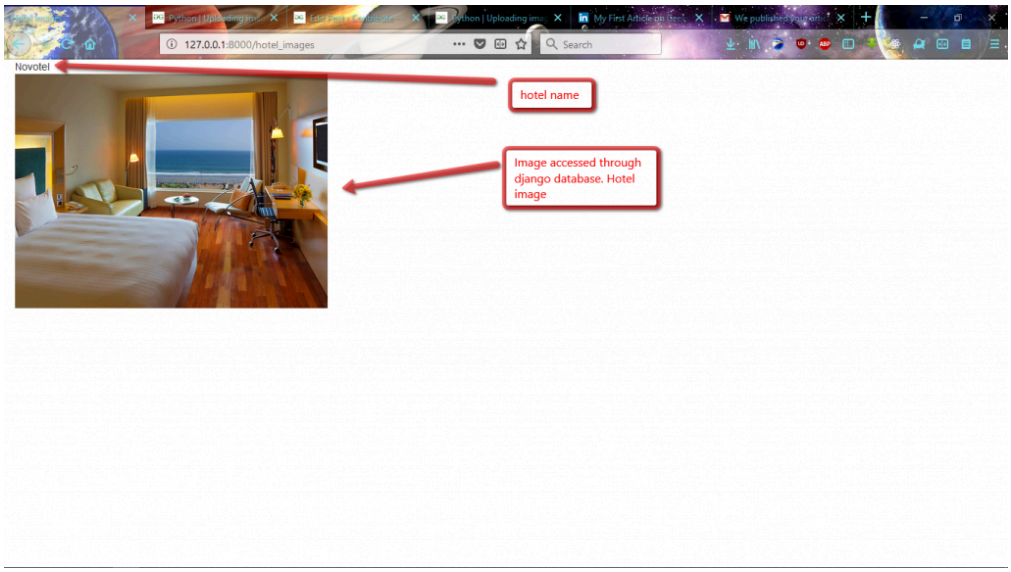
```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Hotel Images</title>
```

```
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css">
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js">
    </script>
    <script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/js/bootstrap.min.js">
    </script>
</head>
<body>

    {% for hotel in hotel_images %}
            <div class="col-md-4">
                    {{ hotel.name }}
                    <img src="{{ hotel.hotel_Main_Img.url }}" class="img-responsive" style="width: 100%; float: left; margin-right: 10px;" />
            </div>
    {% endfor %}

</body>
</html>
```

Insert the url path in the urls.py file

```
# urls.py
path('hotel_images', display_hotel_images, name = 'hotel_images'),
```

Here is the final view on the browser when we try to access the image.



Hotel Image

Previous Article

Intermediate fields in Django | Python

Next Article

Change Object Display Name using __str__ function - Django Models | Python

Similar Reads

Uploading Image Using Django with Firebase

Firebase is a product of Google which helps developers to build, manage, and grow their apps easily. It helps developers to build their apps faster and in a mor...

4 min read

Uploading files on Google Drive using Python

In the modern era of the internet, a large number of mundane or work-related tasks are carried out over the internet. Compromise of data stored on a device...

3 min read

Google Cloud Platform - Ways of Uploading Data to GCS

Before you can harness the power of the cloud, to serve your content, you have to get your data into it. In this article, we will look into all the different ways you ca...

3 min read

Downloading and Uploading Files from Internet

The terms downloading and uploading are generally used while browsing the Internet. Receiving data or a file from the Internet on your computer is referred t...

7 min read

Servlet - Uploading File

Servlets are the Java programs that run on the Java-enabled web server or application server. They are used to handle the request obtained from the...

2 min read

Why would $_FILES be empty when uploading files to PHP ?

While uploading the files or writing the code, developers do many mistakes which we cannot figure out most of the time. Some spelling mistakes, some...

3 min read

Uploading a Reverse Shell to a Web Server in Kali Linux

We basically hack the webserver for gaining access to the system. We look into what is inside the web server and we want to have full control of the web serve...

3 min read

Uploading and Downloading Files in Flask

This article will go over how to upload and download files using a Flask database using Python. Basically, we have a section for uploading files where we can...

7 min read

Uploading and Reading a CSV File in Flask

Flask is a flexible, lightweight web-development framework built using python. A Flask application is a Python script that runs on a web server, which listens to...

3 min read

How to Compress Image in Android Before Uploading it to Firebase Storage?

Usually, in an app we make users upload images like profile images or others. But there is a chance that the uploaded image is only for display purposes, that too i...

4 min read

Article Tags : Technical Scripter Technical Scripter 2018