uploads in Django.

1. Prerequisite Knowledge

In the last article on Django Forms, we have seen that to get the form data; we use **request.POST** in the Form object.

Request POST

But to **upload files to Django**, we need to include another attribute **request.FILES** as well because the uploaded files are stored in the attribute **request.FILES** instead of **request.POST.**

Here's what the code will look like:

```
form =ReviewForm(request.POST,request.FILES)
```

Django has separate model fields to handle the different file types – **ImageField and FileField**.

We use the ImageField when we want to upload only image files(.jpg/.jpeg/.png etc.)

To allow file uploads we need to add the following attribute in the **<form>** attribute.

```
enctype ="multipart/form-data"
```

At the end, the form HTML tag should look like this:

```
<form type='post' enctype ="multipart/form-data">
```

2. Modify settings.py to store uploaded files

Now in the **settings.py** add the following lines at the end of the file.

```
MEDIA_URL = '/media/'
```

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

Here:

- **MEDIA_URL:** This mentions the **URL endpoint**. This is the URL the user can go to and upload their files from the browser
- **MEDIA_ROOT:** This we have seen earlier in the Django Templates article under the DIR settings for Templates.

If you don't understand it right now, you will understand it later in the **article**!

The second line tells Django to store all the uploaded files in a folder called **'media'** created in the BASE_DIR, i.e., the project Directory.

We need to create the folder manually so all the uploaded files will be stored in the media folder underlined below:

Django Project Directory

3. Creating the Media Folder in the Django Project.

Now in the project folder, create a new folder with the name **‘media.’**

Media

Once the folder is created, we will move to creating the eBook upload webpage.

Creating an E-Book upload webpage

Now let us make a webpage, in which the clients can upload the pdf file of books they have.

1. Creating an E-Book Model in models.py

In models.py, create a new Django Model “**EBooksModel”** and then add the following code

```
class EBooksModel(models.Model):


    title = models.CharField(max_length = 80)

    pdf = models.FileField(upload_to='pdfs/')


    class Meta:

        ordering = ['title']


    def __str__(self):

        return f"{self.title}"
```

Here:

- We have used the well-known model **CharField**, which will store the name of the pdf that the client submits.
- **FileField** is used for files that the client will upload.
- **Upload_to** option specifies the path where the file is going to be stored inside the media. For, e.g., I have used ‘pdfs/,’ which implies that the files will get stored in a folder named pdfs inside the media.
- **Class Meta and def__str__:** we have learned this in Django models article

Note: The upload File won't be saved in the database. Only the instance of the file will be saved there. Hence even if you delete that particular instance, the Uploaded file will still be inside the media folder.

You will know what I meant by an **instance of a file** is later in this article, so hold on !!

2. Creating the UploadBookForm in forms.py

We will now import the EBooksModel into forms.py and then create a new ModelForm "**UploadBookForm.**"

Create the Form using the knowledge we learned in Django Forms

```
class UploadBookForm(forms.ModelForm):

    class Meta:

        model = EBooksModel

        fields = ('title', 'pdf',)
```

3. Creating BookUploadView in views.py

The code here will be similar to the one we wrote in Django Forms. But here, we need to accommodate the uploaded files (placed in **request.FILES instead of request.POST.)**

For that, simply add **request.FILES,** along with the **request.POST** as shown below

```
form = UploadBookForm(request.POST,request.FILES)
```

Therefore the full code will be

```
def BookUploadView(request):

    if request.method == 'POST':

        form = UploadBookForm(request.POST,request.FILES)

        if form.is_valid():

            form.save()
            return HttpResponse('The file is saved')
    else:
        form = UploadBookForm()
        context = {
            'form':form,
        }
    return render(request, 'books_website/UploadBook.html', context)
```

4. Creating the UploadBook.html Template

Now we need to create the **<form>** attribute in the template file.

Hence, create a template file " **UploadBook.html."** and add the following.

```
<form method ='post' enctype ="multipart/form-data">
    {% csrf_token %}
    {{form}}
    <input type="submit" value = "Submit">
</form>
```

Don't forget to add **enctype ="multipart/form-data"** otherwise, the form won't work.

Now finally let's map the View with a URL(**book/upload)**

5. Creating a URL path for UploadBookView

Now in the urls.py, add the path to link UploadBookView to **'book/upload.'** using the method we saw in Django URL mapping.

```
path('book/upload', BookUploadView, name ='BookUploadView')
```

Now that we have created a new model, we must perform the migrations again. So in the python shell enter the following command one by one.

```
python manage.py makemigrations
python manage.py migrate
```

That’s it, Now lets run the server and check the browser.

Browser

Voila, the upload form is up !! Now choose a pdf and click the submit button.

Browser

When you hit the submit button, then **“the file has been saved”** page will appear

Browser

If you go to the media folder, you will see a **pdfs** folder and in it the pdf that you submitted.

Media/pdfs

Register the newly made model in the admin site, using:

```
admin.site.register(EBooksModel)
```

Then load the admin site in the browser and go to **EBooksModel** and select the element we just submitted.

Admin Site

Now here, if you observe, in the **pdf field**. You will see a **Currently:** option.

The path that is written in front of it**: pdfs/cprogramming_tutorial.pdf** is called an instance. Therefore **pdfs/<pdf_name>** is an instance of the <pdf_name> file.

Django saves only the instance of the file and not the file itself. Hence even if you delete the model from the admin site, the pdf file will still be there in the media folder.

View the uploaded files from the browser front-end

In the above webpage, the **instance appears as a link.** But if you click on it, you will get an error message.

This happens because the endpoint is not mapped.

Error Message

Now to correct this error, we need to map this endpoint to the particular file. To do that, go to urls.py and add

```
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)
```

Urls

If you read the line, you will get a rough idea of what we are doing

Here:

- In **settings.py**, we have already set **debug = True**, so settings.DEBUG will always be **true**.
- In side **if function**, the following code will add **static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)** to the urlpatterns present above.

Debug 1

The line **static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)** can be thought of in this way.

To the host website(*http://127.0.0.1:8000/*) is where we are adding the endpoints –

- **MEDIA_URL**(which we kept as **'/media/'** in the starting of this article)

Media Settings

- and then **document_root**(which is the **location of the pdf file inside the media folder.**

Hence if I want to view the **cprogramming_tutorial.pdf** file that I earlier uploaded, I will go to *http://127.0.0.1:8000/media/pdfs/cprogramming_tutorial.pdf* (observe how MEDIA_URL(**'/media/'**) is being used)

This is what the above code in **urls.py** does.

That's it, now if you reload the server and click on the instance we saw earlier on the Django admin page, you won't get the error now.

Admin Site

Now click on the instance link and check !!

Pdf Browser

Hence we can now see the pdf via the browser!