Welcome to Our Django Image Upload Tutorial!

Are you ready to take your Django skills to the next level? In this comprehensive tutorial, we'll guide you through the process of uploading images in your Django web application. Whether you're a beginner or an experienced developer, we've got you covered!

What You'll Learn:

- Setting up your Django project for image handling.
- Creating a user-friendly image upload form.
- Handling image uploads in views and models.
- Storing images in your media directory.
- Displaying uploaded images on your web pages.
- Implementing image resizing and optimization.
- And much more!

1. Configuration of PIL for image display.

- First and foremost, you need to have PIL or pillow installed on your computer.
- From your terminal after activating your virtualenv and going into your project folder you run a command **python –m pip install Pillow.**
- For you to see that **PIL** has been downloaded successfully run this command from the terminal **pip freeze.**
- This enables you to see all the downloaded apps in your Django environment, but it doesn't mean that PIL is appended in the installed apps.
- Therefore, you're supposed to go to settings and append it by **'PIL',.**

Note: You should write it like that one above or else you can get errors in your terminal.

2. Configuration of our settings.

```
MEDIA_URL = '/media/'

# is a setting that defines the base URL where media files are served from.
# It specifies the URL prefix that will be used to construct the URLs for accessing media files.


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

# is a setting that defines the absolute filesystem path to the directory where media
# files will be stored on the server's file system.
```

- We go to our settings so that we can put there paths for the server to access our images and to configure our media settings to specify where our uploaded files (including images) will be stored.
- This sets up the URL path and the local filesystem path where your uploaded images will be stored.

3. Configuring our models.py file.

→ We go to models.py to create a model that includes a 'FileField' or 'ImageField' to handle image uploads.

→ The **upload_to** parameter specifies the subdirectory within your **MEDIA_ROOT** where the uploaded images will be stored.

→ We also import PIL package so that we can be able to use the image field by **from PIL import Image**

→

→

4. Configuring our form for image upload.

★ Create a form to handle image uploads. You can use Django's **ModelForm** to generate a form based on your model:

★ We can also import our model "MyImage" if you didn't import all (*) from the models.py file.

5. Configuring Urls.py for the project (master urls.py).

❖ During development, you need to configure your project's URLs to serve media files. Add the following to your project's **urls.py.**

```
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)
```

6. Configuring urls.py for the application.

- Configure your URL patterns to route to the view that handles image uploads.
- Configure your URL patterns to route to the view that handles where you will go after the image has been uploaded.

7. Configuring our views and templates.

- Create views and templates to handle image uploads and display them. In your views, you'll process the form and save the uploaded image.

NOTE: After making some adjustments to the models.py don't forget to go to your terminal to makemigrations and to migrate. Other than that, you will not get the desired output.

Sure, here's a sample documentation on how to delete an image using Django's built-in functionality. You can adapt and expand upon this template to create comprehensive documentation for your project.

✓ Deleting an Image in Your Django Application

In your Django application, you might need to provide users with the ability to delete images. This guide outlines the steps required to implement image deletion functionality using Django's ORM and views.

♦ Prerequisites

Before proceeding, ensure the following:

- You have a working Django project with the necessary app containing your image model.

- You are familiar with creating Django models, views, and templates.

♦ Step 1: Create a Confirmation Page

1. In your `templates` directory, create a new template file named `delete_image.html`.

2. Add the HTML code for the confirmation page. This page allows users to confirm their intention to delete an image. Include a form with a "Delete" button and a "Cancel" link.

```html
<!DOCTYPE html>
<html>
<head>
    <title>Delete Image</title>
</head>
<body>
    <h1>Delete Image</h1>
    <p>Are you sure you want to delete this image?</p>
    <form method="post">
```

```
        {% csrf_token %}
        <input type="submit" value="Delete">
        <a href="{% url 'image_list' %}">Cancel</a>
    </form>
</body>
</html>
```

♦ Step 2: Create a Delete View

1. In your app's `views.py`, create a view function to handle the deletion confirmation page.

```python
from django.shortcuts import render, get_object_or_404, redirect
from myapp.models import MyImage
```

```
from myapp.utils import delete_image


def delete_image_confirmation(request, image_id):
    image = get_object_or_404(MyImage, id=image_id)
    if request.method == 'POST':
        if delete_image(image_id):
            return redirect('image_list')  # Redirect to the image list after successful deletion
        else:
            # Handle deletion failure
            return render(request, 'delete_failure.html')
    return render(request, 'delete_image.html', {'image': image})
```

- **Step 3: Create URLs for Views**

1. In your app's `urls.py`, map the delete confirmation view to a URL.

```python
from django.urls import path
from myapp.views import delete_image_confirmation

urlpatterns = [
    # Other URL patterns
    path('delete/<int:image_id>/confirm/', delete_image_confirmation, name='delete_image_confirmation'),
]
```

♦ Step 4: Update the Image List Template

1. In your app's `templates` directory, update the image list template (`image_list.html`) to include a "Delete" link for each image**.**

```html
<ul>
    {% for image in images %}
    <li>
        {{ image.title }} - <a href="{% url 'delete_image_confirmation' image.id %}">Delete</a>
    </li>
    {% endfor %}
</ul>
```

- **Step 5: Test and Verify**

1. Run your Django development server and navigate to the image list page (`/images/`).

2. Click the "Delete" link next to an image. This will take you to the confirmation page.

3. On the confirmation page, you can confirm the deletion or cancel the operation.

♦ Conclusion

By following these steps, you have successfully implemented image deletion functionality in your Django application. Users can now delete images by confirming their choice on the deletion confirmation page.
