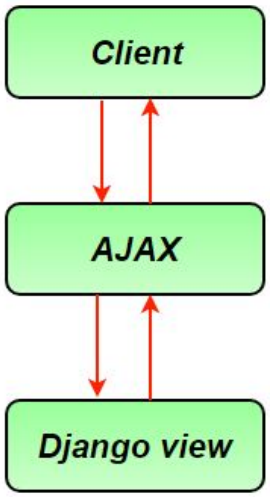
Handling Ajax request in Django

Last Updated : 06 May, 2022



Introduction

This tutorial explains how to carry out an ajax request in the Django web framework. We will create a simple post-liking app as a part of the example.

Glossary

- Project Initialization
- Create models
- Create views
- Write URLs
- Carry out a request with Jquery Ajax.
- Register models to admin and add some posts.

Implementation:

1. Initiate the Django Project – Here I am assuming that you are done with Django Installation.

- To Create a Django Project execute:

```
$ django-admin startproject django_example
```

- After creating a project we need to create a Django app. To create an app say “post” execute the following:

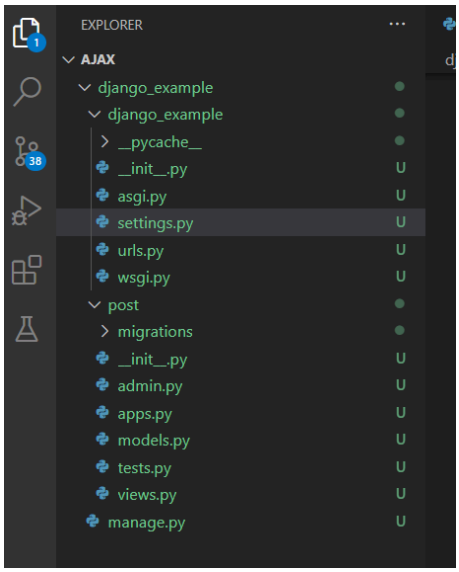
```
$ cd django_example
$ python manage.py startapp post
```

- Go to django_example/settings.py add the post-app

```
# Application definition

INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'post.apps.PostConfig',   # Defining our app
]
```

- Now you will have files something like this:



2. Create models: To create models, go to the post directory and open models.py.

- In models.py. First, create a post table. To create a post table you'll need to write:

```
class Post(models.Model):
    post_heading = models.CharField(max_length=200)
    post_text = models.TextField()
        def __unicode__(self):      # If python2 use __str__ if python3
                return unicode(self.post_heading)
```

- Then In models.py, create like a table. To create like a table you'll need to write:

```
class Like(models.Model):
    post = models.ForeignKey(Post, on_delete = models.CASCADE)
```

- Make Migration and migrate step:

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

After completing these steps, we have our database tables ready to use.

3. Create Views:

To create views, we need to go to the post directory and open views.py

- First, import Previously created Models and HTTP response

```
from .models import Post, Like
from django.http import HttpResponse
```

- Create an index view to render all the posts. Code sample:

```
def index(request):
    posts = Post.objects.all()  # Getting all the posts from database
```

```
    return render(request, 'post/index.html', { 'posts': posts })
```

- Create like posts view to like a post. This view will be called when we will hit a “like button”. Code sample:

```
def likePost(request):
    if request.method == 'GET':
           post_id = request.GET['post_id']
           likedpost = Post.objects.get(pk=post_id) #getting the liked
posts
           m = Like(post=likedpost) # Creating Like Object
           m.save()  # saving it to store in database
           return HttpResponse("Success!") # Sending an success response
    else:
           return HttpResponse("Request method is not a GET")
```

Once our view gets created we will move to write a template and jQuery to perform an ajax request.

4. Create URLs:

To create URLs, open django_example/urls.py. Your django_example/urls.py should look something like this:

```
from django.conf.urls import include, url
from django.contrib import admin
urlpatterns = [
     url(r'^admin/', admin.site.urls),
     url(r'^', include('post.urls')),   # To make post app available at
/
   ]
```

To create URLs, create file post/urls.py. Your post/urls.py should look something like this:

```
from django.conf.urls import url
from . import views
urlpatterns = [
```



```
          url(r'^likepost/$', views.likePost, name='likepost'),   #
```

```
likepost view at /likepost
   ]
```

5. Making templates and carrying out ajax requests:

- Create a file post/templates/post/index.html. Code sample:

```
<!DOCTYPE html>
<html>
<head>
    <title>Like Post App</title>
</head>
<body>
    <p id="message"></p>
    {% for post in posts %}
    <h3>{{ forloop.counter }}) {{ post.post_heading }}</h3>
    <p>{{ post.post_text }} </p>
    <a class="likebutton" id="like{{post.id}}" href="#" data-catid="{{
post.id }}">Like</a>
    {% endfor %}
    <script
src="https://ajax.googleapis.com/ajax/libs/jquery/1.12.0/jquery.min.js">
</script>
    <script type="text/javascript">
    $('.likebutton').click(function(){
    var catid;
    catid = $(this).attr("data-catid");
    $.ajax(
    {
        type:"GET",
        url: "/likepost",
        data:{
                 post_id: catid
        },
        success: function( data )
        {
            $( '#like'+ catid ).remove();
            $( '#message' ).text(data);
        }
     })
```

```
});
</script>
</body>
</html>
```

Basically, what we are doing here is - we are making an ajax get request -> /likepost?post_id=<id_of_liked_post>

6. To Register models to admin and add some posts:

- Open post/admin.py.
- Import Models to admin.py.

```
from .models import Post, Like
```

- Register your models:

```
admin.site.register(Post)
admin.site.register(Like)
```

Now add some posts using the Django default admin portal. Visit http://localhost:8000/ to view and like posts. I also have added this sample app to my github which you may use for reference.

This blog is written by Anshul Singhal.

Are you ready to elevate your web development skills from foundational knowledge to advanced expertise? Explore our Mastering Django Framework - Beginner to Advanced Course on GeeksforGeeks, designed for aspiring developers and experienced programmers. This comprehensive course covers everything you need to know about Django, from the basics to advanced features. Gain practical experience through **hands-on projects** and real-world applications, mastering essential Django principles and techniques. Whether you're just starting or looking to refine your skills, this course will empower you to build sophisticated web applications efficiently. Ready to enhance your web development journey? Enroll now and unlock your potential with Django!

Previous Article

Python | Django Admin Interface

Next Article

Python | User groups with Custom permissions in Django

Similar Reads

What is the Best AJAX Library for Django?

AJAX (Asynchronous JavaScript and XML) is a useful technique that helps web applications update sections of a page without having to reload the whole thing...

4 min read

How To Integrate Ajax with Django Applications

Django is one of the most popular web frameworks for building robust and scalable web applications. However, many modern applications require...

5 min read

Django Request and Response cycle - HttpRequest and HttpResponse...

Prerequisite - Views In Django | PythonBefore we jump into using convenience methods that views come with, Let's talk about the Request and Response cycle...

4 min read

How to get GET request values in Django?

Django, a high-level Python web framework, simplifies the process of web development. One common task in web applications is handling GET requests,...

2 min read

Get Request.User in Django-Rest-Framework Serializer

In Django Rest Framework (DRF), serializers are used to transform complex data types, such as queryset and model instances, into native Python data types. One...

5 min read

How to get JSON data from request in Django?

Handling incoming JSON data in Django is a common task when building web applications. Whether you're developing an API or a web service, it's crucial to...

2 min read

Pass Request Context to Serializer from ViewSet in Django Rest Framework

Django Rest Framework (DRF) is a powerful toolkit for building Web APIs in Django. One of the features that make DRF so flexible is its ability to pass conte...

4 min read

Python Django Handling Custom Error Page

To handle error reporting in Django, you can utilize Django's built-in form validation mechanisms and Django's error handling capabilities. In this article, I'l...

4 min read

Handling Django SECRET_KEY ImproperlyConfigured

In this article, we will go through the common reasons and solutions for the "django core exceptions Improperly Configured" error in Django project. First, let...

3 min read

Adding Tags Using Django-Taggit in Django Project

Django-Taggit is a Django application which is used to add tags to blogs, articles etc. It makes very easy for us to make adding the tags functionality to our djang...

2 min read

Article Tags : Project Python Python Django

Practice Tags : python