Python | User groups with Custom permissions in Django

Last Updated : 31 May, 2022

Let's consider a trip booking service, how they work with different plans and packages. There is a list of product which subscriber gets on subscribing to different packages, provided by the company. Generally, the idea they follow is the level-wise distribution of different products.

Let's see the different packages available on tour booking service :

1. **Starter plan** : In this package, subscriber will get the facility of non-AC bus travel and 1-day stay in a non-AC room only. Let's say the trip is from Delhi to Haridwar(a religious place in Uttarakhand).
2. **Golden Plan** : It will be somewhat costly than the Starter Plan. In this plan, subscriber will be given 2-day stay in a non-AC room, travelling in a AC bus and the trip will be from Delhi to Haridwar, Rishikesh and Mussoorie.
3. **Diamond Plan**: This is the most costly plan, in which subscriber will be provided 3-day plan with AC bus and AC room stay, along with the trip to Haridwar, Rishikesh and Mussoorie and also trip to the Water Park.

Our main objective is to design and write code for the back-end in a very efficient way(following the DRY Principle).
There are multiple methods of implementing this in Django but the most suitable and efficient method is Grouping the Users and defining the permissions of these groups. User of that particular group will automatically inherit the permission of that particular group. Let's define the User model first :

Create a Django application **users**. In models.py file, under 'users' app directory, write this code.

Python3

```
# importing necessary django classes
from django.contrib.auth.models import AbstractUser
from django.utils import timezone
from django.db import models

# User class
class User(AbstractUser):

    # Define the extra fields
    # related to User here
    first_name = models.CharField(_('First Name of User'),
                          blank = True, max_length = 20)

    last_name = models.CharField(_('Last Name of User'),
                          blank = True, max_length = 20)

# More User fields according to need

    # define the custom permissions
    # related to User.
    class Meta:

        permissions = (
            ("can_go_in_non_ac_bus", "To provide non-AC Bus facility"),
            ("can_go_in_ac_bus", "To provide AC-Bus facility"),
            ("can_stay_ac-room", "To provide staying at AC room"),
            ("can_stay_ac-room", "To provide staying at Non-AC room"),
            ("can_go_dehradoon", "Trip to Dehradoon"),
            ("can_go_mussoorie", "Trip to Mussoorie"),
            ("can_go_haridwaar", "Trip to Haridwaar"),
            ("can_go_rishikesh", "Trip to Rishikesh"),

# Add other custom permissions according to need.
```

After migrating the models written above, we have two option for making the group.

1. **Django Admin Panel** : In Admin Panel you will see **Group** in bold letter, Click on that and make 3-different group named level0, level1, level3 . Also, define the custom permissions according to the need.
2. **By Programmatically creating a group with permissions**: Open python shell using python manage.py shell.

Python3

```
# importing group class from django
from django.contrib.auth.models import Group, Permission
from django.contrib.contenttypes.models import ContentType

# import User model
from users.models import User

new_group, created = Group.objects.get_or_create(name ='new_group')

# Code to add permission to group
ct = ContentType.objects.get_for_model(User)

# If I want to add 'Can go Haridwar' permission to level0 ?
permission = Permission.objects.create(codename ='can_go_haridwar',
                                       name ='Can go to Haridwar',
                                               content_type = ct)
new_group.permissions.add(permission)
```

We will set different set of permissions in the same way to all the three groups. Until then, we have made groups and linked it with custom permissions.

Now, check that a particular user is accessing the appropriate functionality like, put a limit that *level0* does not access the functionalities of *level1 users* or *level2 user* and so on. To do this, check the permission on every view function made.
To be very careful here, for the function based view we will simply use the custom decorator.

For example :

Python

```
@group_required('level0')
def my_view(request):
    ...
```

For more details, refer this.

Things get a bit complex when we talk about class-based views, we can not simply just add a decorator function, but we have to make a permission-mixing class.

For example :

Python

```
class GroupRequiredMixin(object):
    ..............
    ....Class Definition.....


class DemoView(GroupRequiredMixin, View):
  group_required = [u'admin', u'manager']

  # View code...
```

For more details, refer this.

References :

1. https://docs.djangoproject.com/en/1.11/topics/class-based-views/mixins/
2. http://bradmontgomery.blogspot.in/2009/04/restricting-access-by-group-in-django.html
3. https://simpleisbetterthancomplex.com/2015/12/07/working-with-django-view-decorators.html
4. https://micropyramid.com/blog/custom-decorators-to-check-user-roles-and-permissions-in-django/

Previous Article

Handling Ajax request in Django

Next Article

Python | Django Admin Interface

Similar Reads

Creating Custom Decorator in Django for different permissions

Decorators are incredibly helpful tools that allow you to dynamically change the functionality of a function, method, or class without having to utilize subclasses...

9 min read

Customizing Object Level Permissions - Django REST Framework

In this article, we will discuss how to customize Object Level Permissions in Django REST Framework. To customize permission classes in Django REST...

5 min read

How To Use Django Permissions Without Defining a Content Type or Model

Django's permissions framework is a powerful feature that allows developers to control access to different parts of their applications. Typically, permissions are...

5 min read

Creating custom user model API extending AbstractUser in Django

Every new Django project should use a custom user model. The official Django documentation says it is “highly recommended” but I’ll go a step further and say...

4 min read

Grant MySQL table and column permissions using Python

MySQL server is an open-source relational database management system that is a major support for web-based applications. Databases and related tables are t...

2 min read

Granting Permissions to Roles in Cassandra

In this article, we are going to discuss how we can granting permission to roles in Cassandra. First, we will create a new role and show how it can access the...

2 min read

How to Solve "File writing permissions blocked by the EPERM" issue in…

Encountering the “EPERM: operation not permitted” error in Node.js when trying to write to a file is a common issue that often indicates insufficient permissions....

4 min read

How to set up file permissions for Laravel?

Setting up proper file permissions for a Laravel application is crucial for its security and functionality. Laravel, a popular PHP framework, often requires the...

4 min read

Permissions And Roles in GitLab

One of the important settings of GitLab is its permission and role system, which decides what users can and cannot do within a project or group. Understanding...

5 min read

How to Change File Permissions in WordPress?

WordPress is a CMS or content management system that is very popular and used because of its features to create and manage websites such as blogs, e-...

5 min read

Article Tags : Experiences GBlog Internship Project +4 More

Practice Tags : python