In our upper code/project

Issue is that anyone come even customer and enter :

seller/dashboard

so they can easily land in seller dashboard that is not good so in thos we can go and fix that

now for that we make custom decorater

**Restrict View based on User Role in Django 5**

# 📘 Restricting Views Based on User Roles in Django 5

In our project, we had an issue where even a **Customer** could manually type the URL /seller/dashboard and access the **Seller Dashboard**.  
This is a **security problem** because different roles (Customer, Seller) should only be allowed to access their respective pages.

To solve this, we created a **custom decorator** that restricts views based on user roles.

## 1. Create a New App: core

We will place all **shared utilities** (like decorators) in a new app named **core**.

Run this command:

python manage.py startapp core

Inside the core app, create a new file called **decorators.py**.

## 2. Create Custom Decorator (core/decorators.py)

# core/decorators.py

# Import necessary utilities

from django.contrib.auth.decorators import login\_required

from functools import wraps

from django.http import HttpResponseForbidden

def login\_and\_role\_required(required\_role):

    """

    Custom decorator to restrict views by login + role.

    - First ensures the user is logged in (via @login\_required).

    - Then checks the role: only 'customer' or 'seller' can access.

    - If role mismatch: returns 403 Forbidden.

    """

    def decorator(view\_func):

        @wraps(view\_func)   # Preserve original function name for debugging

        @login\_required     # User must be logged in

        def \_wrapped\_view(request, \*args, \*\*kwargs):

            user = request.user

            # If view requires 'customer' role but user is not a customer

            if required\_role == "customer" and not user.is\_customer:

                return HttpResponseForbidden("You are not Authorized to access this page ")

            # If view requires 'seller' role but user is not a seller

            if required\_role == "seller" and not user.is\_seller:

                return HttpResponseForbidden("You are not Authorized to access this page ")

            # If role is correct → allow access

            return view\_func(request, \*args, \*\*kwargs)

        return \_wrapped\_view

    return decorator

## 3. Update customer/views.py

Before:  
We were using only @login\_required, which allowed any logged-in user (customer or seller) to access the view.  
Now:  
We replace it with our custom @login\_and\_role\_required("customer").

# customer/views.py

from django.shortcuts import render, redirect

from django.contrib.auth.forms import PasswordChangeForm

from django.contrib import messages

from django.contrib.auth import logout

from core.decorators import login\_and\_role\_required

@login\_and\_role\_required("customer")

def customer\_dashboard\_view(request):

    """

    Dashboard for customers only.

    Only logged-in customers can access this page.

    """

    return render(request, 'customer/dashboard.html')

@login\_and\_role\_required("customer")

def password\_change\_view(request):

    """

    Password change view for customers only.

    Uses Django's PasswordChangeForm.

    After password change, the user is logged out for security.

    """

    if request.method == "POST":

        form = PasswordChangeForm(user=request.user, data=request.POST)

        if form.is\_valid():

            form.save()       # Updates password in the database

            logout(request)   # Force logout -> user must re-login

            messages.success(request, "Password changed successfully. Please log in with your new password.")

            return redirect('login')

        # Optional: Show field errors as flash messages

        # else:

        #     for field, errors in form.errors.items():

        #         for error in errors:

        #             messages.error(request, error)

    else:

        form = PasswordChangeForm(user=request.user)

    return render(request, 'customer/password\_change.html', {'form': form})

## 4. Update seller/views.py

# seller/views.py

from django.shortcuts import render

from core.decorators import login\_and\_role\_required

@login\_and\_role\_required("seller")

def seller\_dashboard\_view(request):

    """

    Dashboard for sellers only.

    Only logged-in sellers can access this page.

    """

    return render(request, 'seller/dashboard.html')

## 5. Update base.html

We also updated the navbar so that:

* If a **Customer** logs in → only the **Customer Dashboard** button is shown.
* If a **Seller** logs in → both **Customer Dashboard** and **Seller Dashboard** buttons are shown (in case the seller has dual roles).
* Guests (not logged in) see only **Register** and **Login** buttons.

{% if user.is\_authenticated %}

    {% if user.is\_customer %}

        <li><a href="{% url 'customer\_dashboard' %}">Customer Dashboard</a></li>

    {% endif %}

    {% if user.is\_seller %}

        <li><a href="{% url 'seller\_dashboard' %}">Seller Dashboard</a></li>

    {% endif %}

    <li>

        <form method="post" action="{% url 'logout' %}">

            {% csrf\_token %}

            <button type="submit">Logout</button>

        </form>

    </li>

{% else %}

    <li><a href="{% url 'register' %}">Register</a></li>

    <li><a href="{% url 'login' %}">Login</a></li>

{% endif %}

# ✅ Final Result

* Customers can **only** access /customer/dashboard (not /seller/dashboard).
* Sellers can **only** access /seller/dashboard.
* Unauthorized access will return **403 Forbidden**.
* Navbar dynamically shows dashboard buttons based on the user’s role.