Here’s the **project structure** for your fully featured Django e-commerce application myshop:

myshop/

├── manage.py

├── db.sqlite3

├── media/ # Uploaded product images

├── static/ # Static files (CSS, JS, etc.)

├── myshop/ # Project settings

│ ├── \_\_init\_\_.py

│ ├── settings.py

│ ├── urls.py

│ └── wsgi.py

├── catalog/ # Main app for products and orders

│ ├── \_\_init\_\_.py

│ ├── admin.py

│ ├── models.py # Category, Product, CartItem, Order, OrderItem

│ ├── views.py # All views including index, category, cart, checkout

│ ├── urls.py

│ ├── migrations/

│ └── templates/

│ └── catalog/

│ ├── index.html # Welcome page with categories

│ ├── category\_page.html # Category-specific product listings

│ ├── product\_detail.html # Product detail with image

│ ├── cart.html # Shopping cart with quantity update

│ ├── checkout.html # Checkout form and order summary

│ └── order\_confirmation.html # Confirmation with reference number

├── accounts/ # Custom user model and RBAC

│ ├── \_\_init\_\_.py

│ ├── admin.py

│ ├── models.py # CustomUser with role field

│ └── migrations/

**✅ Key Features Included**

* **Custom User Model** with roles: Admin, Manager, Staff, Customer
* **Product Categories**: Meat, Egg, Cassava Processed, Fruits & Juices Transformed, Vegetables
* **Product Detail Page** with image upload
* **Shopping Cart** with quantity updates
* **Checkout Page** with order summary and personal data
* **Order Confirmation Page** with unique reference number
* **Email Notifications** for order confirmation
* **Custom Admin Dashboard** with filters and search

Would you like me to walk you through customizing the admin dashboard further (e.g., inline editing, dashboard stats), or help you deploy this structure to GitHub or a live server?

The code execution for generating role-based dashboards and customizing the admin interface failed, but I’ll guide you through implementing it manually. Here's how to structure **dashboard views per role**, **customize the admin**, and **implement role-based access control (RBAC)** in your Django project.

**🔐 Role-Based Access Control (RBAC) — Overview**

RBAC ensures that users can only access views and data appropriate to their role. In your project, roles are defined in the CustomUser model:

class CustomUser(AbstractUser):

ROLE\_CHOICES = (

('admin', 'Admin'),

('manager', 'Manager'),

('staff', 'Staff'),

('customer', 'Customer'),

)

role = models.CharField(max\_length=20, choices=ROLE\_CHOICES, default='customer')

**🧭 Role-Based Dashboard Views**

**✅ Create a dashboard view in catalog/views.py:**

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

from django.shortcuts import render

@login\_required

def dashboard(request):

role = request.user.role

context = {'role': role}

if role == 'admin':

context['admin\_data'] = 'Admin dashboard content here'

return render(request, 'catalog/dashboard\_admin.html', context)

elif role == 'manager':

context['manager\_data'] = 'Manager dashboard content here'

return render(request, 'catalog/dashboard\_manager.html', context)

elif role == 'staff':

context['staff\_data'] = 'Staff dashboard content here'

return render(request, 'catalog/dashboard\_staff.html', context)

else:

context['customer\_data'] = 'Customer dashboard content here'

return render(request, 'catalog/dashboard\_customer.html', context)

**✅ Add to urls.py:**

path('dashboard/', views.dashboard, name='dashboard'),

**✅ Create templates:**

* dashboard\_admin.html
* dashboard\_manager.html
* dashboard\_staff.html
* dashboard\_customer.html

Each template can show relevant data, links, and actions based on the role.

**🛠️ Customize Admin by Role**

**✅ Use ModelAdmin and override get\_queryset:**

class ProductAdmin(admin.ModelAdmin):

list\_display = ('name', 'category', 'price', 'stock')

def get\_queryset(self, request):

qs = super().get\_queryset(request)

if request.user.role == 'manager':

return qs.filter(category\_\_name\_\_in=['Meat', 'Egg']) # Example

elif request.user.role == 'staff':

return qs.none()

return qs

You can also override has\_change\_permission, has\_view\_permission, etc., to restrict actions.

**🧱 Decorator for Role-Based Access**

**✅ Create a reusable decorator:**

from django.contrib.auth.decorators import user\_passes\_test

def role\_required(role):

def check(user):

return user.is\_authenticated and user.role == role

return user\_passes\_test(check)

**✅ Use it in views:**

@role\_required('manager')

def manage\_inventory(request):

...

Would you like me to regenerate the ZIP with these RBAC features and dashboards included, or help you design the dashboard UI for each role?

The ZIP generation with role-based dashboards and RBAC decorators was interrupted, but I’ll guide you through the setup manually and help you design the dashboard UI for each role.

**🔐 Role-Based Access Control (RBAC) Decorator**

**✅ Decorator Definition (accounts/decorators.py)**

from django.contrib.auth.decorators import user\_passes\_test

def role\_required(role):

def check(user):

return user.is\_authenticated and user.role == role

return user\_passes\_test(check)

**✅ Usage in Views**

@role\_required('manager')

def manager\_dashboard(request):

...

You can also create a multi-role decorator:

def roles\_required(\*roles):

def check(user):

return user.is\_authenticated and user.role in roles

return user\_passes\_test(check)

**🧭 Dashboard Views per Role**

**✅ View Logic (catalog/views.py)**

@login\_required

def dashboard(request):

role = request.user.role

if role == 'admin':

return render(request, 'catalog/dashboard\_admin.html')

elif role == 'manager':

return render(request, 'catalog/dashboard\_manager.html')

elif role == 'staff':

return render(request, 'catalog/dashboard\_staff.html')

else:

return render(request, 'catalog/dashboard\_customer.html')

**✅ URL Mapping (catalog/urls.py)**

path('dashboard/', views.dashboard, name='dashboard'),

**🎨 Dashboard UI Design Suggestions**

**🛠 Admin Dashboard (dashboard\_admin.html)**

* Overview of all users and roles
* Product and category management
* Order tracking and status updates
* System logs or analytics

**📦 Manager Dashboard (dashboard\_manager.html)**

* Inventory overview
* Product performance charts
* Order summaries
* Notifications for low stock

**👷 Staff Dashboard (dashboard\_staff.html)**

* Daily task list
* Order fulfillment queue
* Product packaging status

**🛒 Customer Dashboard (dashboard\_customer.html)**

* Order history
* Track current orders
* Recommended products
* Account settings

Would you like me to retry generating the ZIP file with these dashboards and decorators included, or help you build the HTML templates for each dashboard role?

The ZIP regeneration with role-based dashboards and decorators failed again, but I’ll help you implement everything manually and clearly.

**🔐 RBAC Decorators**

**✅ role\_required — Single Role**

from django.contrib.auth.decorators import user\_passes\_test

def role\_required(role):

def check(user):

return user.is\_authenticated and user.role == role

return user\_passes\_test(check)

**✅ roles\_required — Multiple Roles**

def roles\_required(\*roles):

def check(user):

return user.is\_authenticated and user.role in roles

return user\_passes\_test(check)

**✅ Usage in Views**

@role\_required('manager')

def manager\_dashboard(request):

...

@roles\_required('admin', 'manager')

def shared\_dashboard(request):

...

**🧭 Dashboard Views per Role**

**✅ View Logic (catalog/views.py)**

from django.shortcuts import render

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

@login\_required

def dashboard(request):

role = request.user.role

if role == 'admin':

return render(request, 'catalog/dashboard\_admin.html')

elif role == 'manager':

return render(request, 'catalog/dashboard\_manager.html')

elif role == 'staff':

return render(request, 'catalog/dashboard\_staff.html')

else:

return render(request, 'catalog/dashboard\_customer.html')

**✅ URL Mapping (catalog/urls.py)**

path('dashboard/', views.dashboard, name='dashboard'),

**🎨 HTML Templates for Dashboards**

**🛠 dashboard\_admin.html**

A black background with a black square

AI-generated content may be incorrect.

**📦 dashboard\_manager.html**

A black background with a black square

AI-generated content may be incorrect.

**👷 dashboard\_staff.html**

A black background with a black square

AI-generated content may be incorrect.

**🛒 dashboard\_customer.html**

A black background with a black square

AI-generated content may be incorrect.

Would you like me to retry generating the ZIP with these templates and decorators included, or help you deploy this version to GitHub or a live server?