



# **MYSORE UNIVERSITY SCHOOL OF ENGINEERING**

Manasagangotri campus, Mysuru-570006  
(Approved by AICTE, New Delhi)



## **UNIVERSITY OF MYSORE**

**An Assignment (21CD71, Full Stack Development) Report**

**On**

**“E-Commerce Product Management System Report”**

### **Submitted By**

**SHREYA B N**

**21SECD40**

7<sup>th</sup> Semester

Department of CS&D

MUSE

### **Under Faculty In charge**

- 1) Dr. M. S. Govinde Gowda, Director
- 2) Mr. Karthik MN  
Asst. Professor  
Dept. of CS&D  
MUSE

# **E-Commerce Product Management System Report**

## **Table of Contents**

<b>1. Introduction.....</b>	<b>Page 1</b>
<b>2. Product and Category Models.....</b>	<b>Page 2</b>
<b>3. One-to-Many Relationship.....</b>	<b>Page 3</b>
<b>4. Django Admin Setup.....</b>	<b>Page 4</b>
<b>5. Displaying Products with ListView.....</b>	<b>Page 5</b>
<b>6. Adding Discount Field via Migrations.....</b>	<b>Page 6 - 7</b>
<b>7. Conclusion.....</b>	<b>Page 8</b>

## 1. Introduction

E-commerce platforms require a structured approach to managing products and categories. This report details the implementation of an **E-Commerce Product Management System** using Django. The system enables efficient product categorization, admin management, product listing with ListView, and schema evolution using Django migrations.

The key objectives of this system are:

- Organizing products under specific categories.
- Establishing a **One-to-Many relationship** between categories and products.
- Managing product data efficiently using Django's built-in admin interface.
- Displaying product lists dynamically using Django's **ListView**.
- Using **Django migrations** to modify the database schema as the system evolves.

This report provides a step-by-step guide to implementing each of these features.

## 2. Product and Category Models

We define two models: Category and Product. Each product belongs to a category, creating a One-to-Many relationship.

```
from django.db import models
```

```
class Category(models.Model):
```

```
    name = models.CharField(max_length=255)
```

```
    def __str__(self):
```

```
        return self.name
```

```
class Product(models.Model):
```

```
    name = models.CharField(max_length=255)
```

```
    description = models.TextField()
```

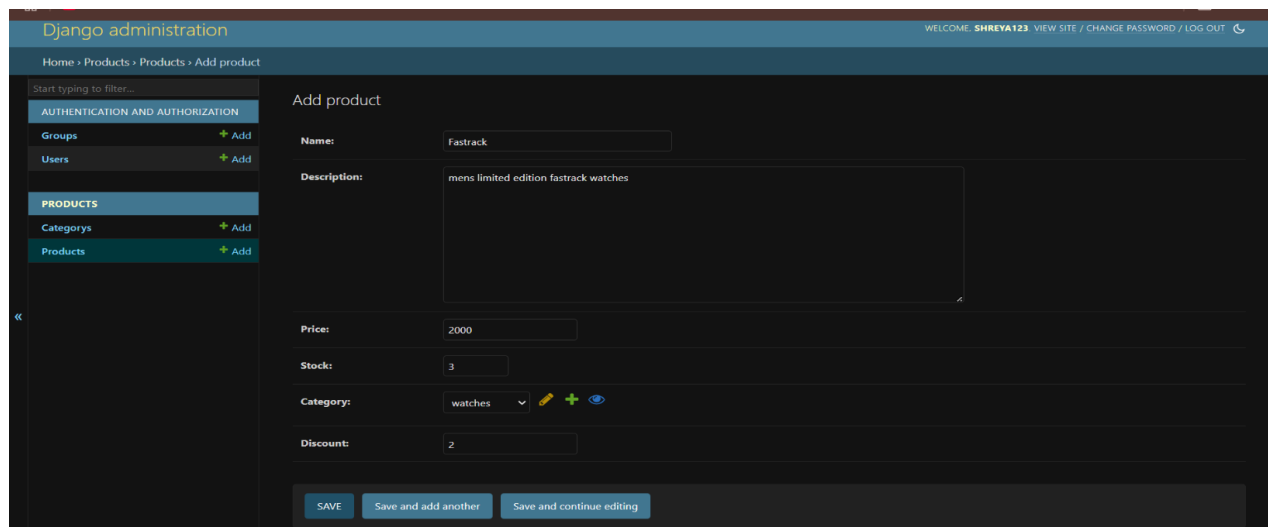
```
    price = models.DecimalField(max_digits=10, decimal_places=2)
```

```
    stock = models.IntegerField()
```

```
    category = models.ForeignKey(Category, on_delete=models.CASCADE)
```

```
    def __str__(self):
```

```
        return self.name
```



The screenshot displays the Django administration interface. The top navigation bar includes the 'Django administration' logo and user information: 'WELCOME SHREYA123 VIEW SITE / CHANGE PASSWORD / LOG OUT'. The breadcrumb trail shows 'Home > Products > Products > Add product'. On the left sidebar, the 'PRODUCTS' section is expanded, showing 'Categories' and 'Products' with '+ Add' links. The main content area is titled 'Add product' and contains a form with the following fields: 'Name' (text input with value 'Fastrack'), 'Description' (text area with value 'mens limited edition fastrack watches'), 'Price' (text input with value '2000'), 'Stocks' (text input with value '3'), 'Category' (dropdown menu with value 'watches' and a '+ Add' link), and 'Discount' (text input with value '2'). At the bottom of the form are three buttons: 'SAVE', 'Save and add another', and 'Save and continue editing'.

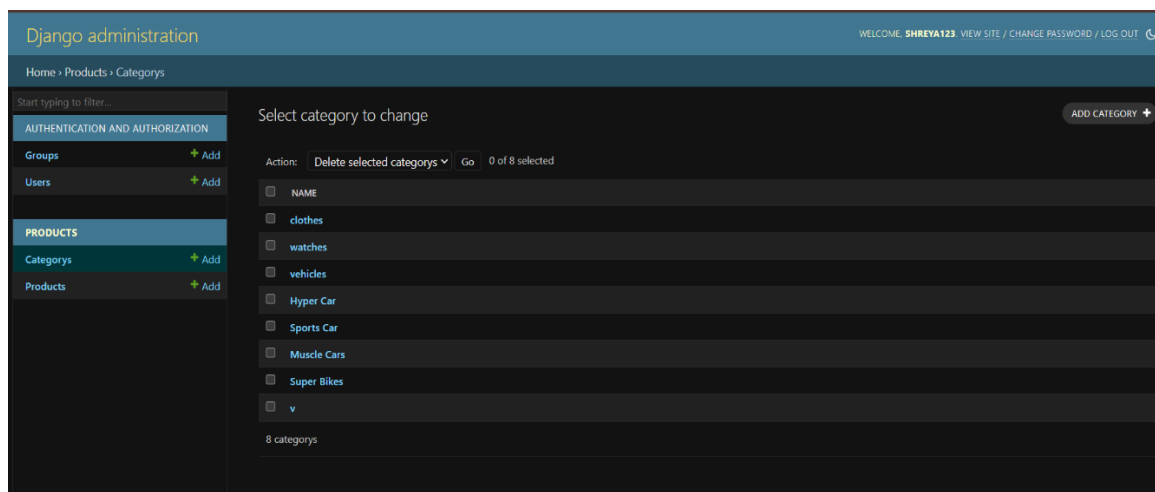
```
admin.py x
products > admin.py
1 from django.contrib import admin
2 from .models import Product, Category
3
4 @admin.register(Category)
5 class CategoryAdmin(admin.ModelAdmin):
6     list_display = ('name',)
7
8 @admin.register(Product)
9 class ProductAdmin(admin.ModelAdmin):
10     list_display = ('name', 'price', 'stock', 'category')
11     list_filter = ('category',)
12     search_fields = ('name', 'description')
13
14
```

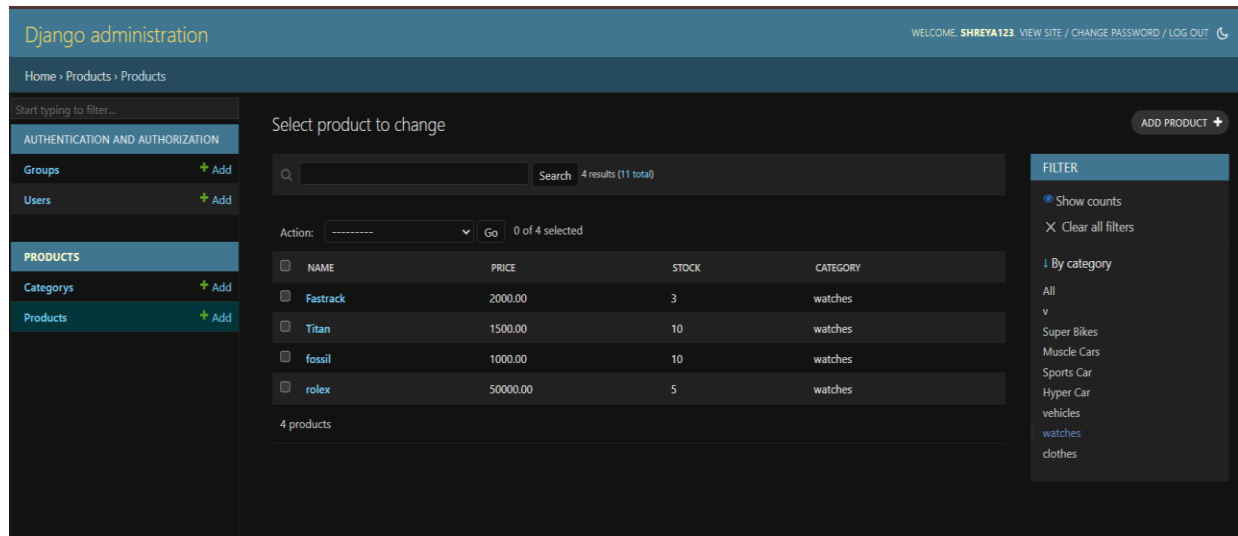
### 3. One-to-Many Relationship

The ForeignKey in Product links each product to a category, enforcing the One-to-Many relationship.

Example:

- Category: Watches
- Products: Rolex, Fossil, Titan, Fastrack (All belong to Watches)





## 4. Django Admin Setup

Django Admin provides a convenient interface to manage products and categories.

To enable this, we register our models in `admin.py`:

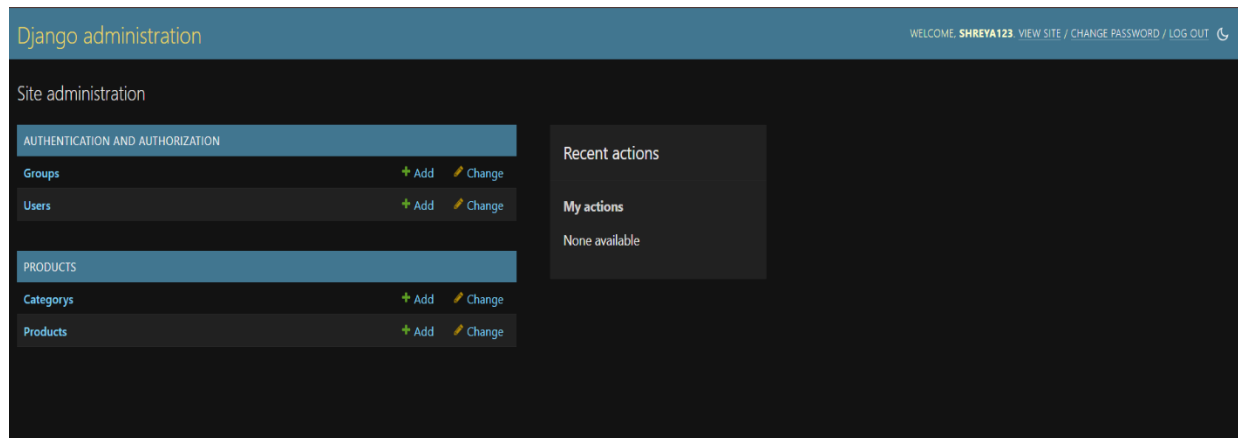
```
from django.contrib import admin
```

```
from .models import Category, Product
```

```
admin.site.register(Category)
```

```
admin.site.register(Product)
```

Once registered, both categories and products appear in the Django Admin dashboard, where they can be added, updated, or deleted with ease.



## 5. Displaying Products with ListView

To display products on the website, we use **Django's ListView**, which simplifies querying and rendering data dynamically.

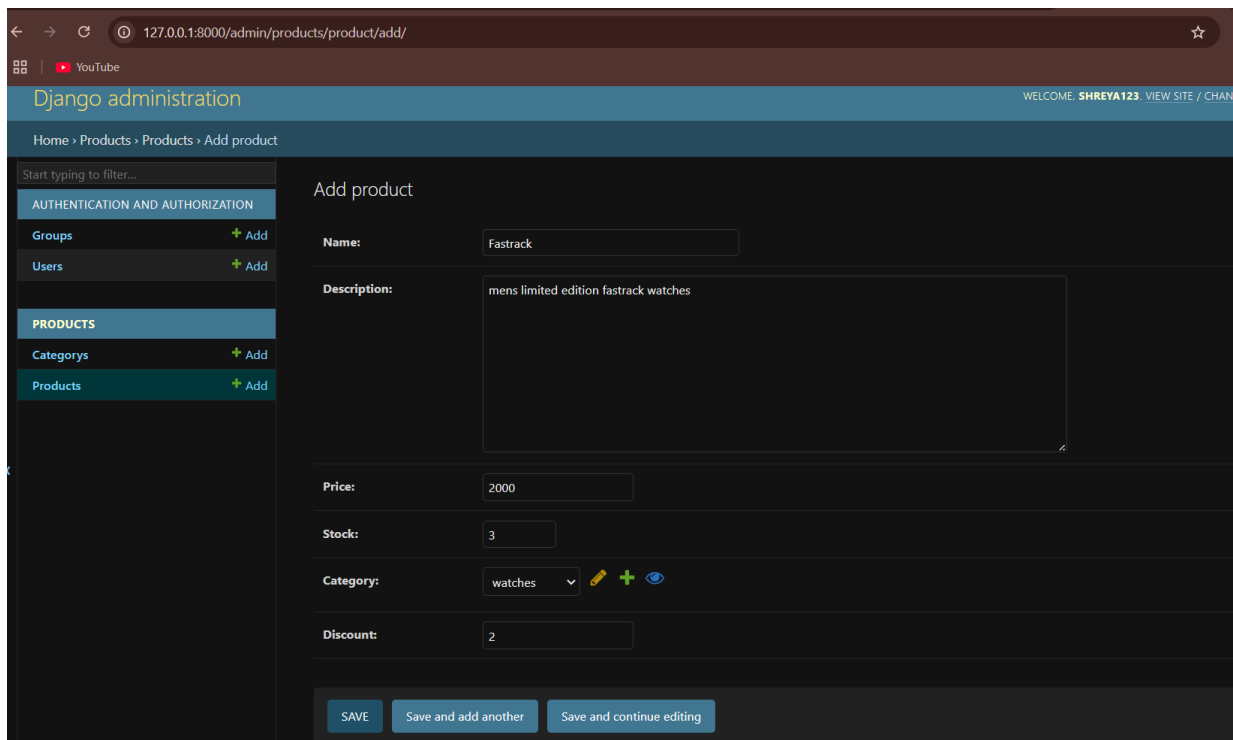
views.py:

```
from django.views.generic import ListView
from .models import Product
class ProductListView(ListView):
    model = Product
    template_name = 'products/product_list.html'
    context_object_name = 'products'
```

product\_list.html:

```
<h1>Product List</h1>
<ul>
    {% for product in products %}
        <li>{{ product.name }} - ${{ product.price }} (Stock: {{ product.stock
    }})</li>
    {% endfor %}
</ul>
```





## 6. Adding Discount Field via Migrations

During development, new fields may be added to the models. Here, we add a discount

field to `Product` using Django migrations

Modify `models.py`:

```
class Product(models.Model):  
    name = models.CharField(max_length=255)  
    description = models.TextField()  
    price = models.DecimalField(max_digits=10, decimal_places=2)  
    stock = models.IntegerField()  
    category = models.ForeignKey(Category, on_delete=models.CASCADE)  
    discount = models.DecimalField(max_digits=5, decimal_places=2, default=0.00)
```

2. Run Migrations:



python manage.py makemigrations

python manage.py migrate

```
models.py 2 X
products > models.py > ...
1  from django.db import models
2
3  class Category(models.Model):
4      name = models.CharField(max_length=255, unique=True)
5
6      def __str__(self):
7          return self.name
8
9  class Product(models.Model):
10     name = models.CharField(max_length=255)
11     description = models.TextField()
12     price = models.DecimalField(max_digits=10, decimal_places=2)
13     stock = models.PositiveIntegerField()
14     category = models.ForeignKey(Category, on_delete=models.CASCADE, related_name="products")
15     discount = models.DecimalField(max_digits=5, decimal_places=2, default=0.00)
16
17
18     def __str__(self):
19         return self.name
20
```

```
11     description = models.TextField()
12     price = models.DecimalField(max_digits=10, decimal_places=2)
13     stock = models.PositiveIntegerField()
14     category = models.ForeignKey(Category, on_delete=models.CASCADE, related_name="products")
15     discount = models.DecimalField(max_digits=5, decimal_places=2, default=0.00)
16
17
18     def __str__(self):
19         return self.name
20
```

PROBLEMS 6 OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\Lap\Desktop\ecommerce\ecommerce> python manage.py makemigrations
>> python manage.py migrate
>>
No changes detected
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, products, sessions
Running migrations:
  No migrations to apply.
PS C:\Users\Lap\Desktop\ecommerce\ecommerce>
```

## 7. Conclusion

This report demonstrated the complete implementation of an E-Commerce Product Management System in Django, covering:

- Database design with One-to-Many relationships
- Django Admin setup for easy management
- Product listing with ListView
- Schema evolution via migrations (Adding discount field)

This system provides a strong foundation for further enhancements like authentication, search filters, and cart functionality.

**GitHub link:** <https://github.com/Shreyadiya12/ecommerce-product-management-system/tree/master>