Objective & Scope of the project

The scope of the project will be limited to some functions of the e-commerce website. It will display products, customers can select catalogs and select products, and can remove products from their cart specifying the quantity of each item. Selected items will be collected in a cart. At checkout, the item on the card will be presented as an order. Customers can pay for the items in the cart to complete an order. This project has great future scope. The project also provides security with the use of login ID and passwords, so that no unauthorized users can access your account. The only authorized person who has the appropriate access authority can access the software.

Theoretical Background of project

This project deals with developing a Virtual website **'E-commerce Website'**. It provides the user with a list of the various products available for purchase in the store. For the convenience of online shopping, a shopping cart is provided to the user. After the selection of the goods, it is sent for the order confirmation process. The system is implemented using Python's web framework Django. To develop an e-commerce website, it is necessary to study and understand many technologies.

System analysis & design

By analyzing the design, we found out there will be two types of requirements for our E-Commerce Store to be working.

Customer Side Design: -

- User should be able to Sign Up and Login as a Customer.
- User Should be able to see his/her previous orders
- The user will be able to browse for the products and add items to cart.
- User can checkout after logging in.

Admin Side Design: -

- Admin should be able to login and accessed the dashboard.
- Admin should have privileges that normal Customers won't have.
- Admin should be able to check all orders status (Pending, Completed).
- Admin should be able to add delete edit Products.
- Admin should be able to add delete edit Customers.
- Admin should be able to add delete edit Orders.
- Admin can change his credential or make new superuser accounts

Methodology adopted; System Implementation & details of Hardware & Software used

Language Used: Python

IDE used: Visual Studio Code is a standalone source code editor that runs on Windows, macOS, and Linux. The top pick for JavaScript and web developers, Python developers with extensions to support just about any programming language.

Packages Used: -

- *Django*: Django is a high-level Python web framework that encourages rapid development and clean, pragmatic design. Built by experienced developers, it takes care of much of the hassle of web development, so you can focus on writing your app without needing to reinvent the wheel. It's free and open source.
- *appdirs*: A small Python module for determining appropriate platform-specific dirs, e.g. a "user data dir".
- *asgiref*: ASGI is a standard for Python asynchronous web apps and servers to communicate with each other, and positioned as an asynchronous successor to WSGI.
- *astroid:* "**asteroid"** is a library for AST parsing, static analysis and inference, currently powering most of **pylint** capabilities. It offers support for parsing Python source code into ASTs, similar to how the builtin **"ast"** module works.
- *Distlib* : Distlib is a library which implements low-level functions that relate to packaging and distribution of Python software. It is intended to be used as the basis for third-party packaging tools.
- *filelock:* A platform independent file lock.
- *lazy-object-proxy:* A fast and thorough lazy object proxy. We use lazy-object-proxy when you only have the object way later and you use wrapt.ObjectProxy when you want to override few methods (by subclassing) and forward everything else to the target object.
- *mccabe:* Ned's script to check McCabe complexity. This module provides a plugin for flake8, the Python code checker.
- *pillow:* The Python Imaging Library adds image processing capabilities to your Python interpreter. This library provides extensive file format support, an efficient internal representation, and fairly powerful image processing capabilities. The core image library is designed for fast access to data stored in a few basic pixel formats. It should provide a solid foundation for a general image processing tool.
- *pylint:* Pylint analyses your code without actually running it. It checks for errors, enforces a coding standard, looks for code smells, and can make suggestions about how the code could be refactored. Pylint can infer actual values from your code using its internal code representation (astroid
- *pytz:* pytz brings the Olson tz database into Python. This library allows accurate and cross platform timezone calculations using Python 2.4 or higher. It also solves the issue of ambiguous times at the end of daylight saving time, which you can read more about in the Python Library
- *sqlparse:* **sqlparse** is a non-validating SQL parser for Python. It provides support for parsing, splitting and formatting SQL statements.

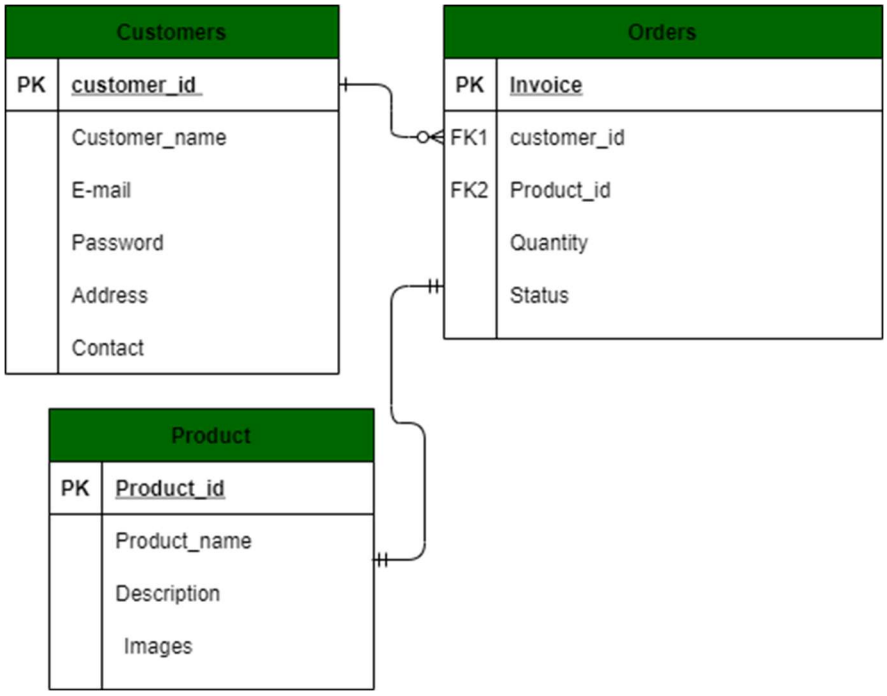
- *wrapt:* The aim of the **wrapt** module is to provide a transparent object proxy for Python, which can be used as the basis for the construction of function wrappers and decorator functions.

Detailed Life Cycle of the Project

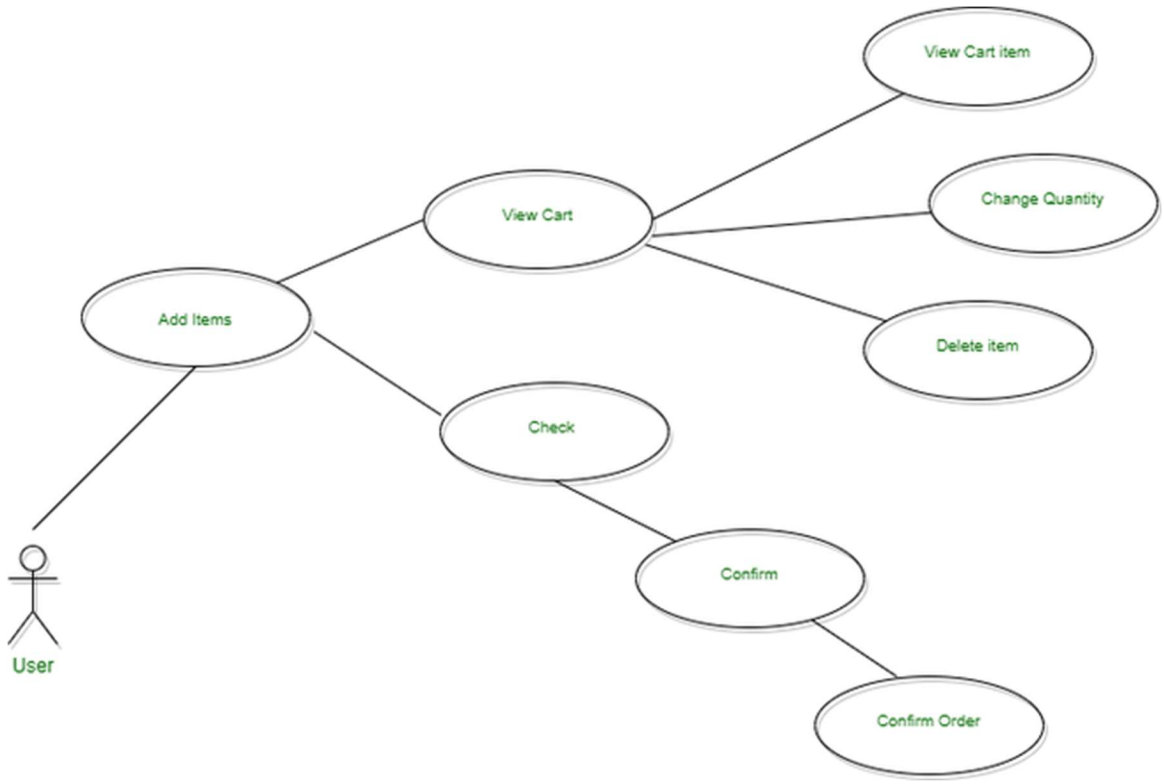
Entity Relationship Diagram for Customer

Customer Interface:

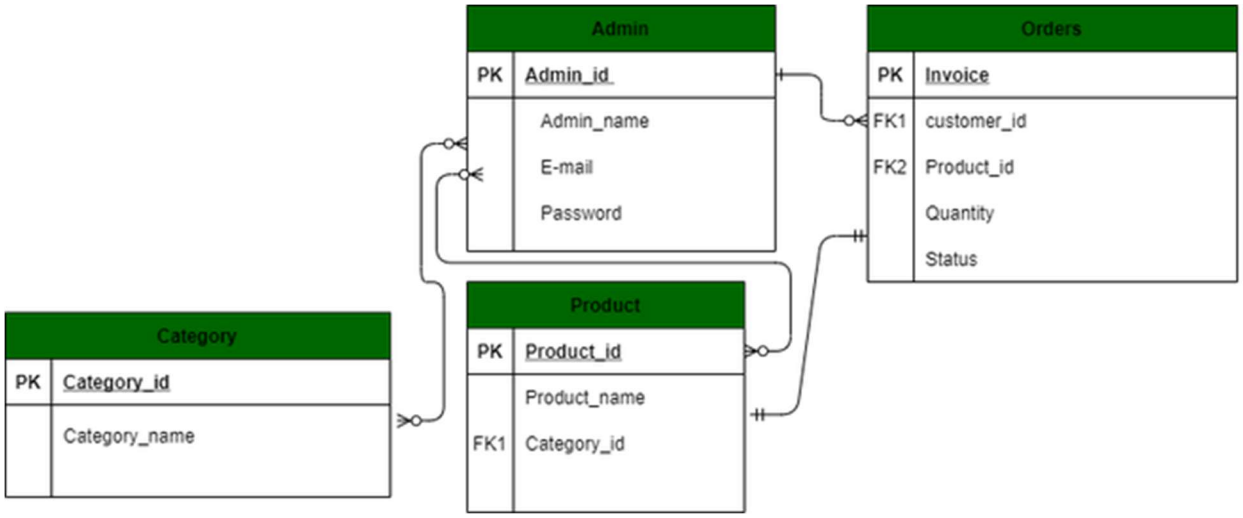
1. Customer shops for a product
2. Customer changes quantity
3. The customer adds an item to the cart
4. Customer views cart
5. Customer checks out
6. Customer sends order



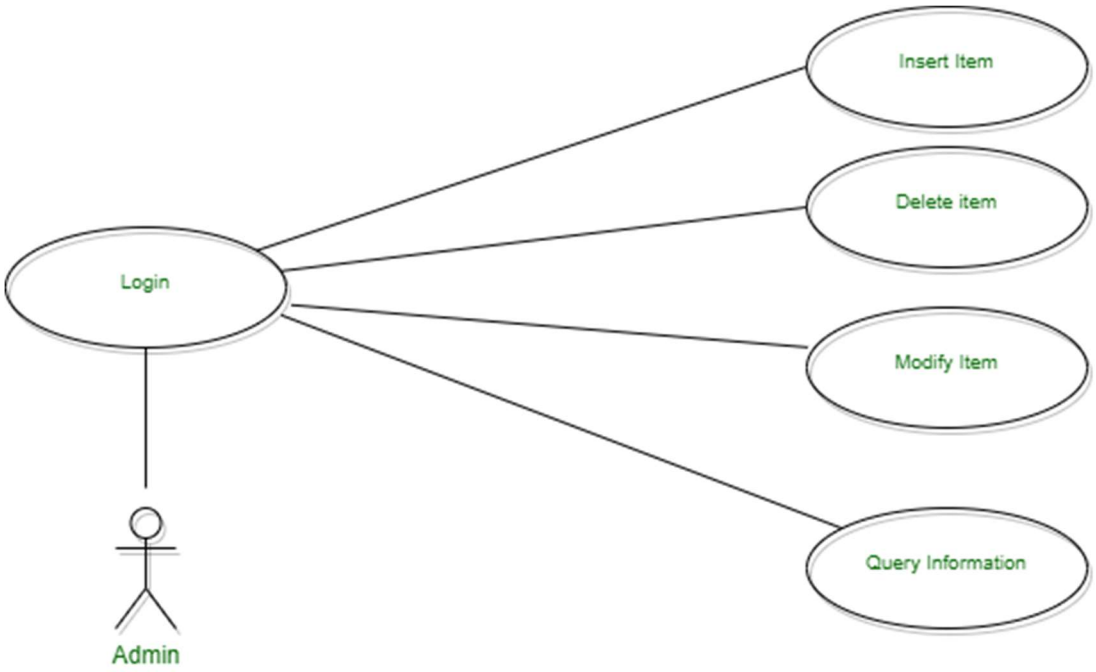
Data Flow Diagram For Customer:



Entity Relationship Diagram for Admin

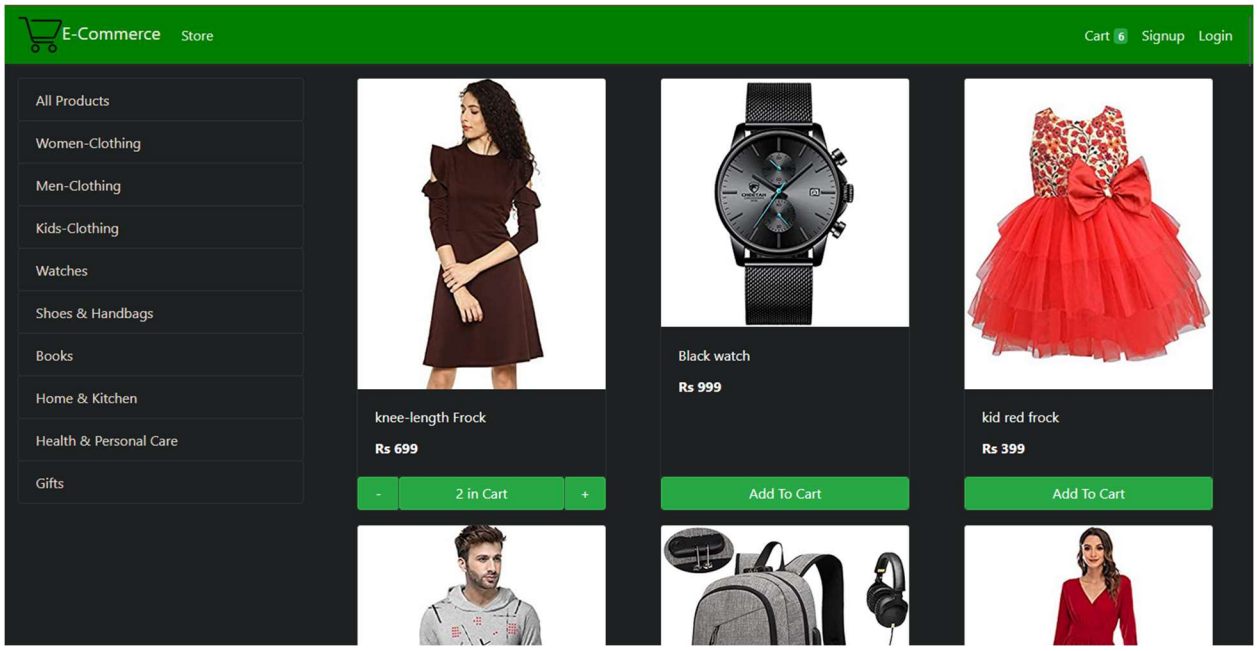


Data Flow Diagram For Admin:

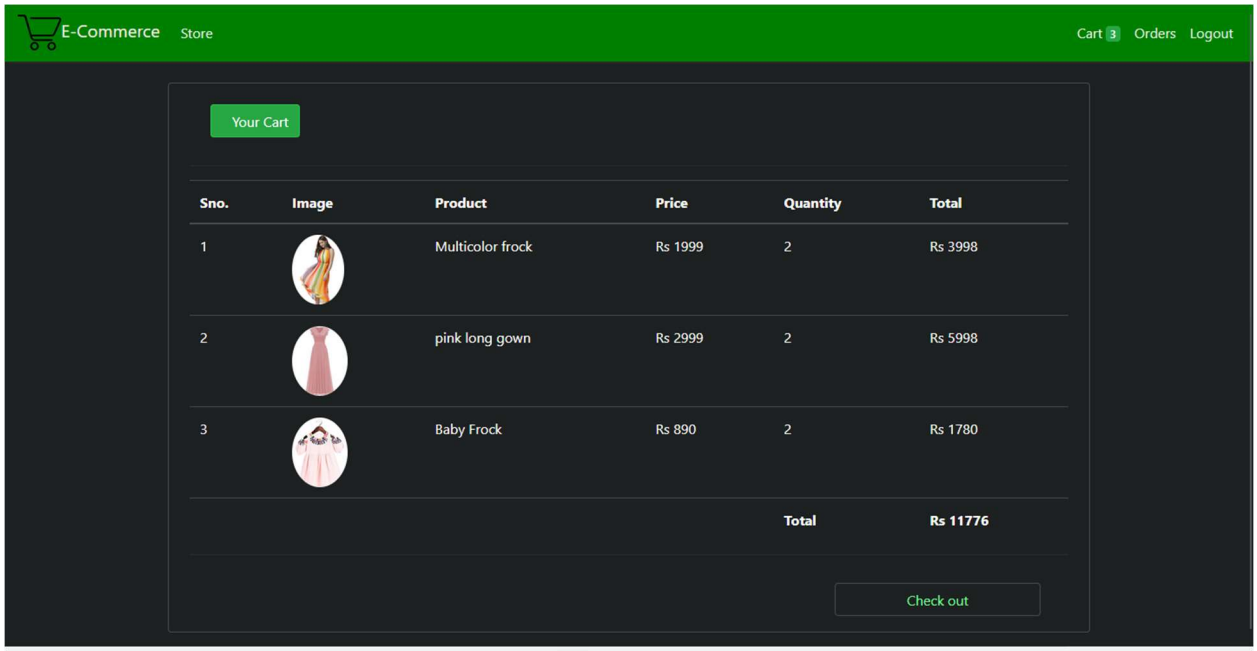


Input and Output screen Design

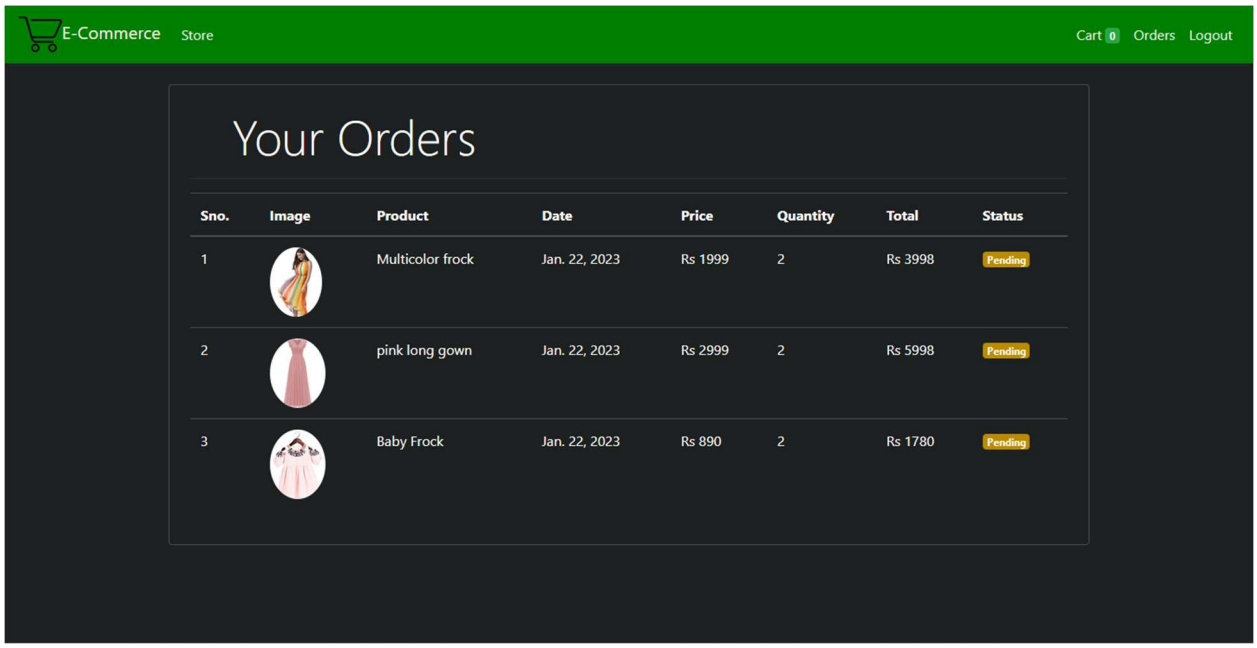
Main Page (Homepage)



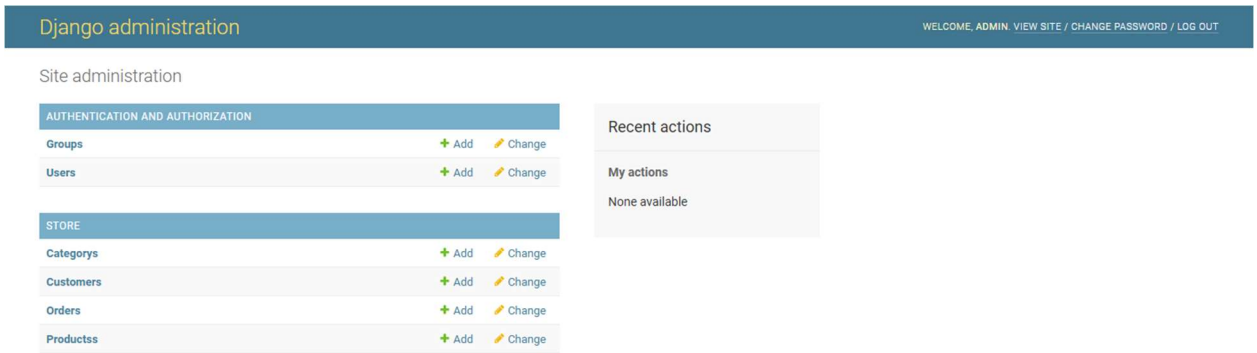
Shopping Cart :



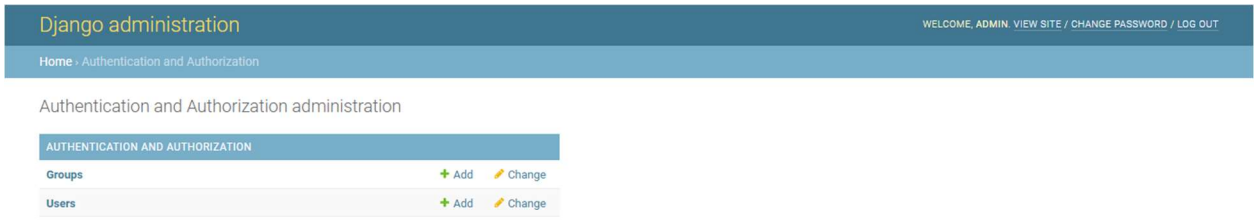
Customer Order:



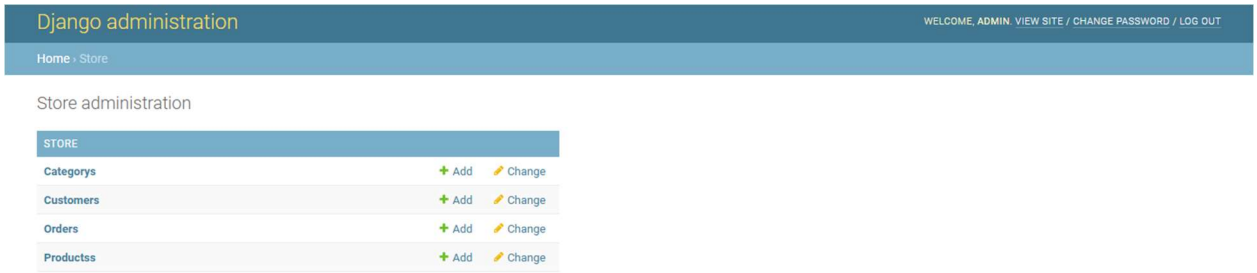
Admin Panel:



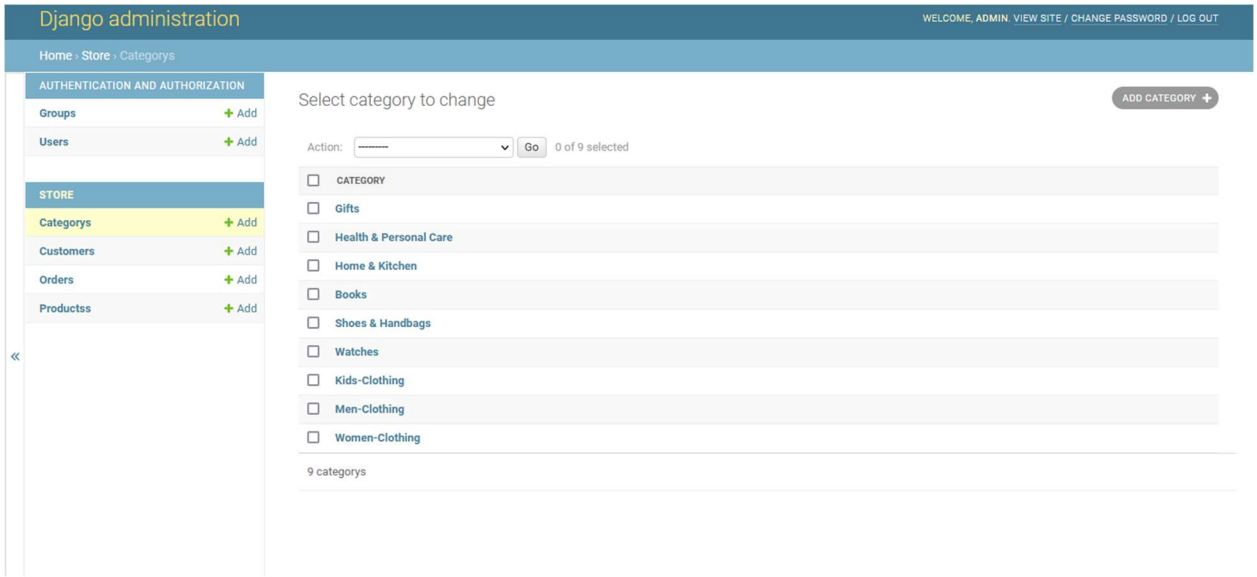
Authentication & Authorization Admin:



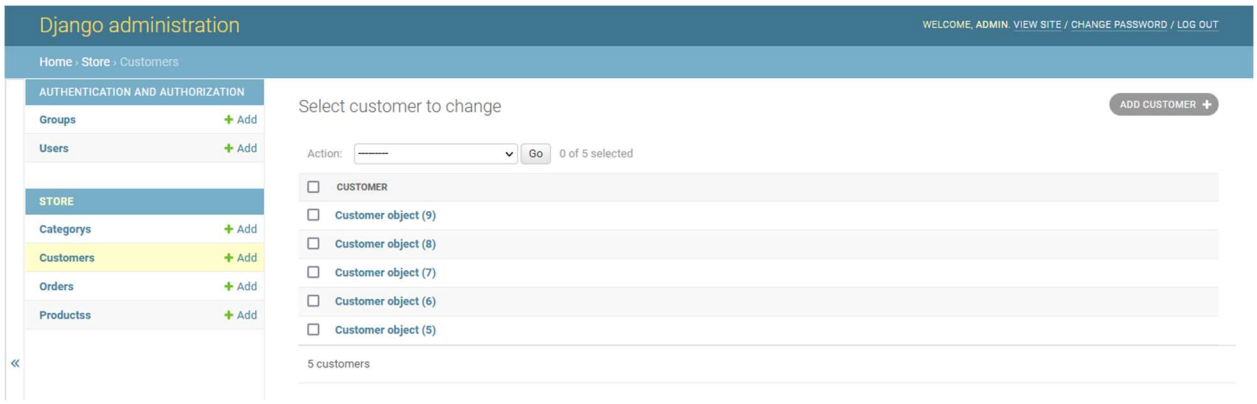
Store Admin:



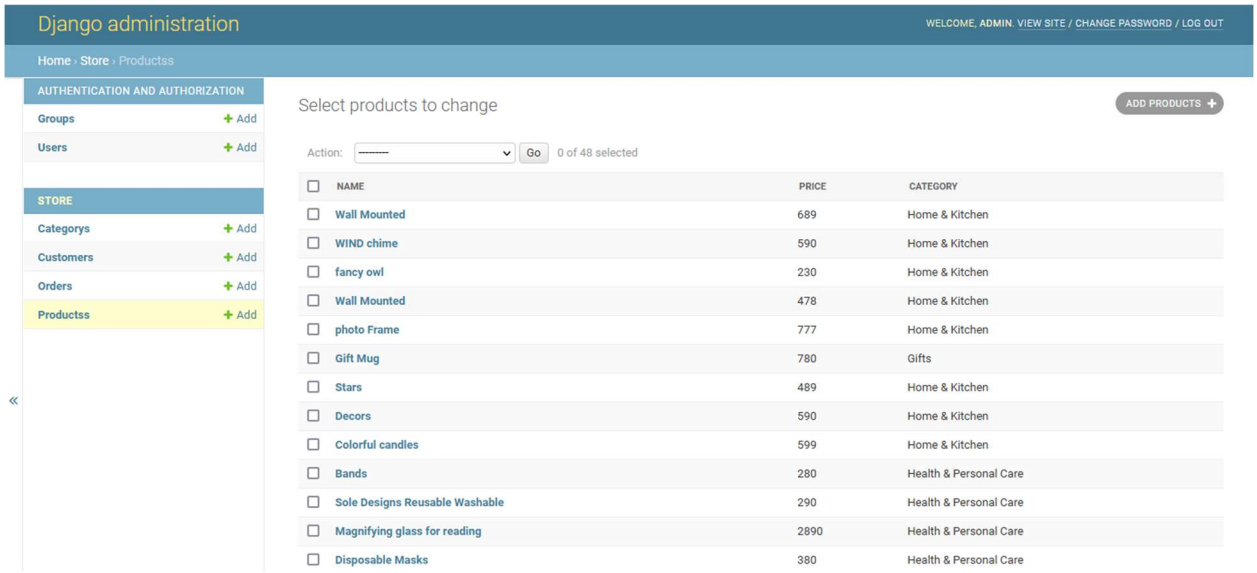
Edit / Add Category:



Adding Customer:



Adding/Editing Products:



Process involved

Design Level

Package: A **python package** is a collection of modules. Modules that are related to each other are mainly put in the same package.

Function: In Python, a function is a group of related statements that performs a specific task. Functions help break our program into smaller and modular chunks. As our program grows larger and larger, functions make it more organized and manageable. Furthermore, it avoids repetition and makes the code reusable.

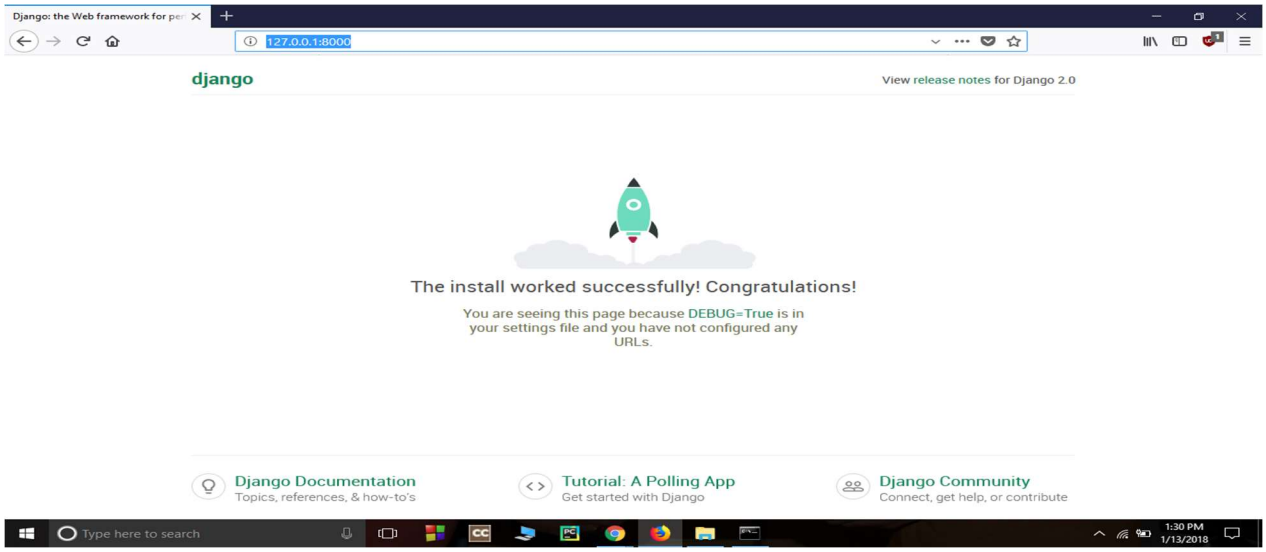
Application Name: E Shop

Building E-Commerce Store in Python

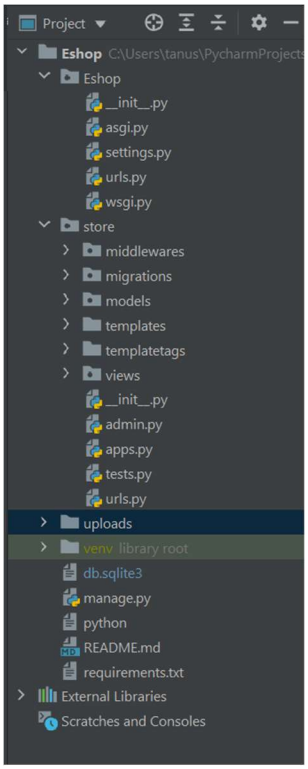
Firstly we will install Django and all the Modules that is required, then we will create Django Project.

When we execute ***django-admin startproject*** command, then it will create a Django project inside the normal project which we already have created here. ***django-admin startproject Eshop.***

Run Default Django webserver:- Django internally provides a default webserver where we can launch our applications. ***python manage.py runserver*** command in terminal**.** By default, the server runs on port 8000. Access the webserver at the highlighted URL.



Open the project folder using a text editor. The directory structure should look like this :



Now add store app in E-commerce website in **settings.py**.

```
# Application definition

INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'store'
]
```

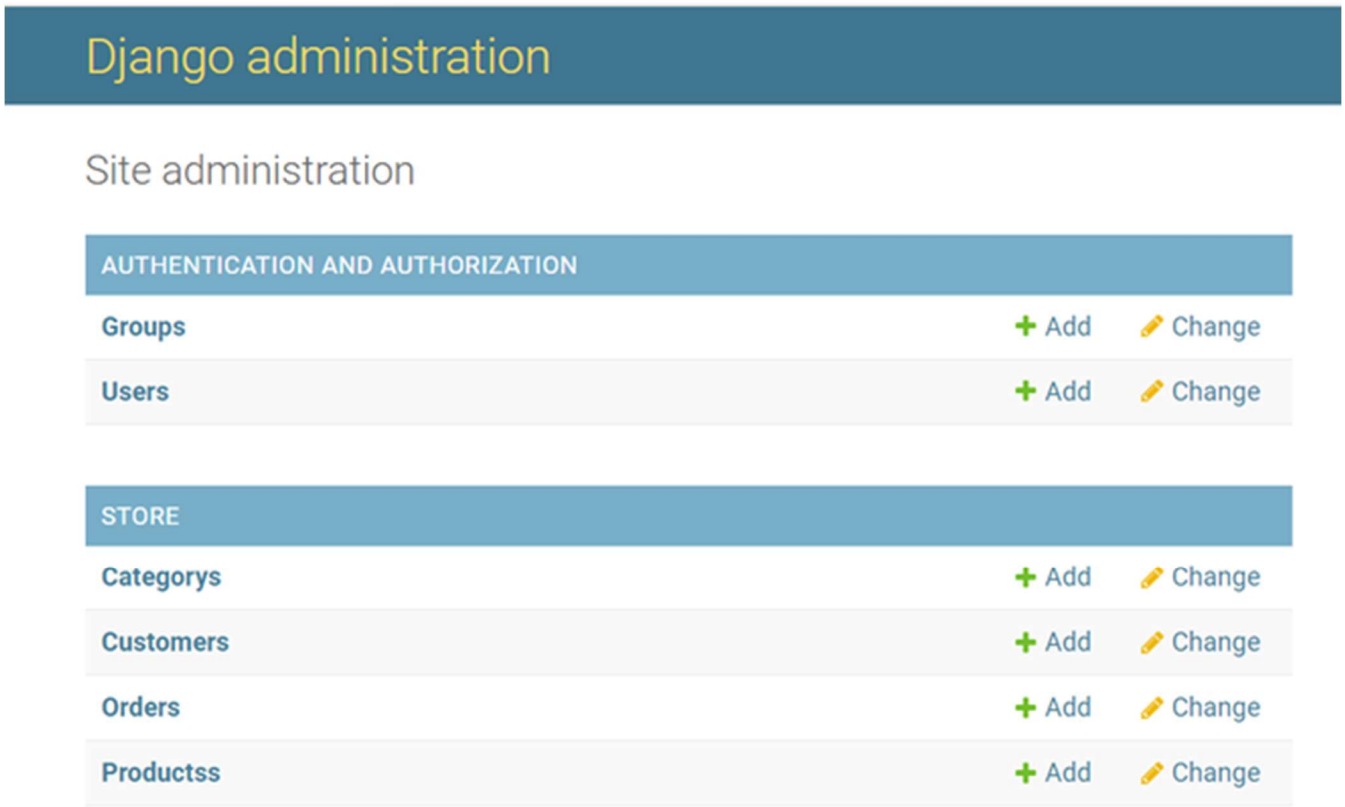
urls.py

This file contains all the URL patterns used by the website

```
from django.contrib import admin
from django.urls import path, include
from django.conf.urls.static import static
from . import settings

urlpatterns = [
    path('admin/', admin.site.urls),
    path('', include('store.urls'))
] + static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)
```

Models

The below screenshot shows the required models that we will need to create. These models are tables that will be stored in the SQLite database.



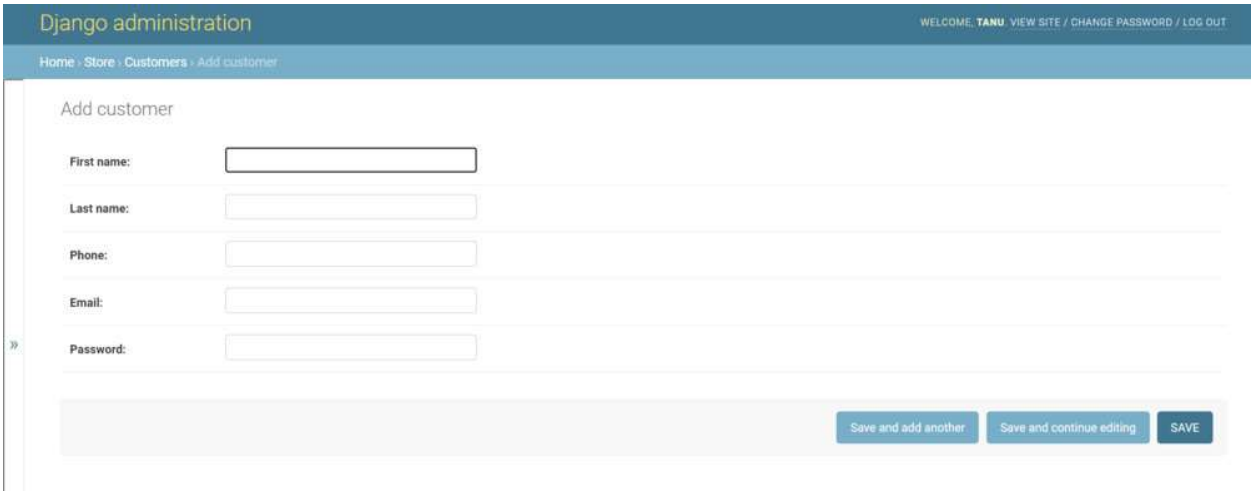
Let’s see each model and the fields required by each model.

category.py

```
from django.db import models

class Category(models.Model):
    name = models.CharField(max_length=50)

    @staticmethod
    def get_all_categories():
        return Category.objects.all()

    def __str__(self):
        return self.name
```

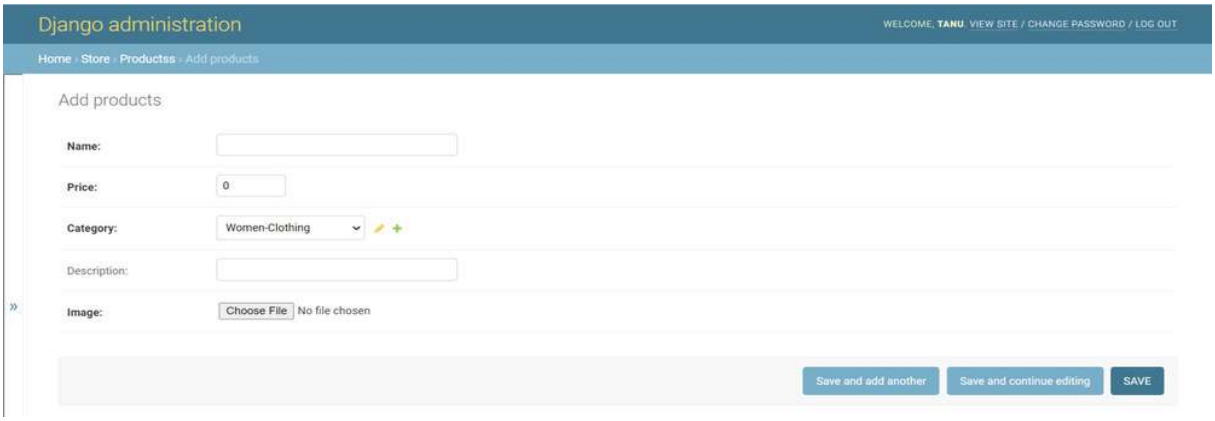
customer.py

```
from django.db import models

class Customer(models.Model):
    first_name = models.CharField(max_length=50)
    last_name = models.CharField(max_length=50)
    phone = models.CharField(max_length=10)
    email = models.EmailField()
    password = models.CharField(max_length=100)

    # to save the data
    def register(self):
        self.save()

    @staticmethod
    def get_customer_by_email(email):
        try:
            return Customer.objects.get(email=email)
        except:
            return False

    def isExists(self):
        if Customer.objects.filter(email=self.email):
            return True

        return False
```



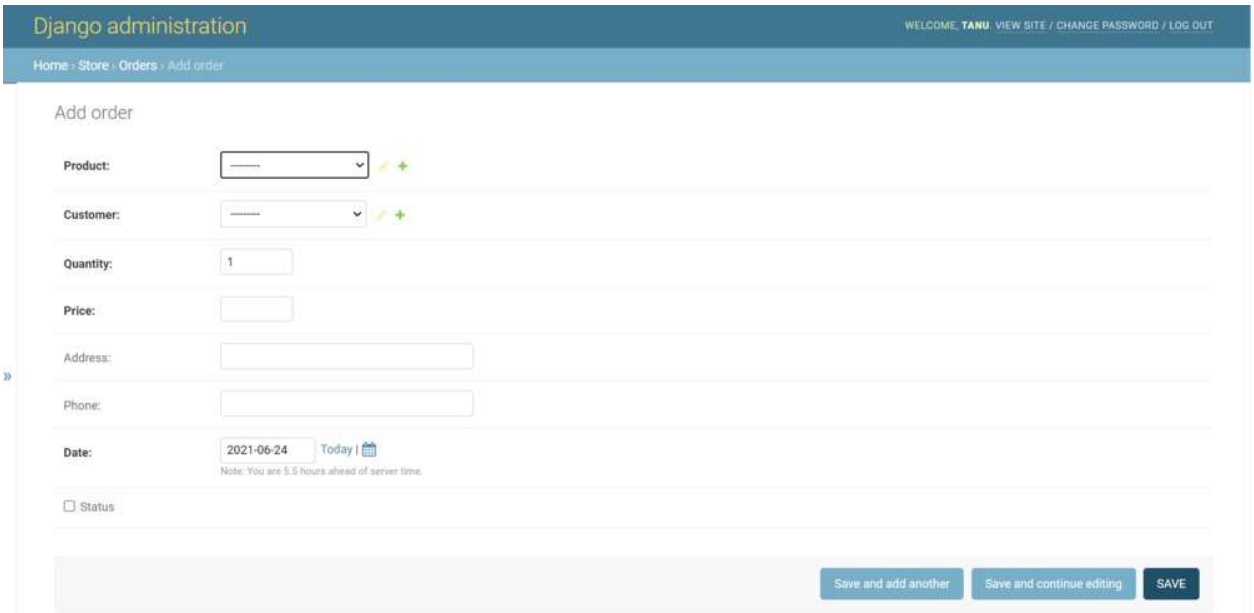
products.py

```
from django.db import models
from .category import Category

class Products(models.Model):
    name = models.CharField(max_length=60)
    price = models.IntegerField(default=0)
    category = models.ForeignKey(Category, on_delete=models.CASCADE, default=1)
    description = models.CharField(
        max_length=250, default='', blank=True, null=True)
    image = models.ImageField(upload_to='uploads/products/')

    @staticmethod
    def get_products_by_id(ids):
        return Products.objects.filter(id__in=ids)

    @staticmethod
    def get_all_products():
        return Products.objects.all()

    @staticmethod
    def get_all_products_by_categoryid(category_id):
        if category_id:
            return Products.objects.filter(category=category_id)
        else:
            return Products.get_all_products()
```



Orders.py

```
from django.db import models
from .product import Products
from .customer import Customer
import datetime

class Order(models.Model):
    product = models.ForeignKey(Products,
                                on_delete=models.CASCADE)
    customer = models.ForeignKey(Customer,
                                 on_delete=models.CASCADE)
    quantity = models.IntegerField(default=1)
    price = models.IntegerField()
    address = models.CharField(max_length=50, default='', blank=True)
    phone = models.CharField(max_length=50, default='', blank=True)
    date = models.DateField(default=datetime.datetime.today)
    status = models.BooleanField(default=False)

    def placeOrder(self):
        self.save()

    @staticmethod
    def get_orders_by_customer(customer_id):
        return Order.objects.filter(customer=customer_id).order_by('-date')
```



Views :

In views, we create a view named *home.py, login.py, signup.py, cart.py, checkout.py, orders.py* which takes a request and renders an HTML as a response. Create an *home.html, login.html, signup.html, cart.html, checkout.html, orders.html* in the templates. And map the views to the store\urls.py folder.

```
from django.contrib import admin
from django.urls import path
from .views.home import Index, store
from .views.signup import Signup
from .views.login import Login, logout
from .views.cart import Cart
from .views.checkout import CheckOut
from .views.orders import OrderView
from .middlewares.auth import auth_middleware


urlpatterns = [
    path('', Index.as_view(), name='homepage'),
    path('store', store, name='store'),

    path('signup', Signup.as_view(), name='signup'),
    path('login', Login.as_view(), name='login'),
    path('logout', logout, name='logout'),
    path('cart', auth_middleware(Cart.as_view()), name='cart'),
    path('check-out', CheckOut.as_view(), name='checkout'),
    path('orders', auth_middleware(OrderView.as_view()), name='orders'),

]
```

The below files show the views for each functionality of the site.

home.py

```
from django.shortcuts import render, redirect, HttpResponseRedirect
from store.models.product import Products
from store.models.category import Category
from django.views import View


# Create your views here.
class Index(View):

    def post(self, request):
        product = request.POST.get('product')
        remove = request.POST.get('remove')
        cart = request.session.get('cart')
        if cart:
            quantity = cart.get(product)
            if quantity:
                if remove:
                    if quantity <= 1:
                        cart.pop(product)
                    else:
                        cart[product] = quantity-1
                else:
                    cart[product] = quantity+1

            else:
                cart[product] = 1
        else:
            cart = {}
            cart[product] = 1

        request.session['cart'] = cart
        print('cart', request.session['cart'])
        return redirect('homepage')

    def get(self, request):
        # print()
        return HttpResponseRedirect(f'/store{request.get_full_path()[1:]}')


def store(request):
    cart = request.session.get('cart')
    if not cart:
        request.session['cart'] = {}
```

```
products = None
categories = Category.get_all_categories()
categoryID = request.GET.get('category')
if categoryID:
    products = Products.get_all_products_by_categoryid(categoryID)
else:
    products = Products.get_all_products()

data = {}
data['products'] = products
data['categories'] = categories

print('you are : ', request.session.get('email'))
return render(request, 'index.html', data)
```

login.py

```
from django.shortcuts import render, redirect, HttpResponseRedirect
from django.contrib.auth.hashers import check_password
from store.models.customer import Customer
from django.views import View


class Login(View):
    return_url = None

    def get(self, request):
        Login.return_url = request.GET.get('return_url')
        return render(request, 'login.html')

    def post(self, request):
        email = request.POST.get('email')
        password = request.POST.get('password')
        customer = Customer.get_customer_by_email(email)
        error_message = None
        if customer:
            flag = check_password(password, customer.password)
            if flag:
                request.session['customer'] = customer.id

                if Login.return_url:
                    return HttpResponseRedirect(Login.return_url)
                else:
                    Login.return_url = None
                    return redirect('homepage')
            else:
                error_message = 'Invalid !!'
        else:
            error_message = 'Invalid !!'

        print(email, password)
        return render(request, 'login.html', {'error': error_message})


def logout(request):
    request.session.clear()
    return redirect('login')
```

signup.py

```
from django.shortcuts import render, redirect
from django.contrib.auth.hashers import make_password
from store.models.customer import Customer
from django.views import View


class Signup (View):
    def get(self, request):
        return render(request, 'signup.html')

    def post(self, request):
        postData = request.POST
        first_name = postData.get('firstname')
        last_name = postData.get('lastname')
        phone = postData.get('phone')
        email = postData.get('email')
        password = postData.get('password')
        # validation
        value = {
            'first_name': first_name,
            'last_name': last_name,
            'phone': phone,
            'email': email
        }
        error_message = None

        customer = Customer(first_name=first_name,
                            last_name=last_name,
                            phone=phone,
                            email=email,
                            password=password)
        error_message = self.validateCustomer(customer)

        if not error_message:
            print(first_name, last_name, phone, email, password)
            customer.password = make_password(customer.password)
            customer.register()
            return redirect('homepage')
        else:
            data = {
                'error': error_message,
                'values': value
```

```
        }
        return render(request, 'signup.html', data)

def validateCustomer(self, customer):
    error_message = None
    if (not customer.first_name):
        error_message = "Please Enter your First Name !!"
    elif len(customer.first_name) < 3:
        error_message = 'First Name must be 3 char long or more'
    elif not customer.last_name:
        error_message = 'Please Enter your Last Name'
    elif len(customer.last_name) < 3:
        error_message = 'Last Name must be 3 char long or more'
    elif not customer.phone:
        error_message = 'Enter your Phone Number'
    elif len(customer.phone) < 10:
        error_message = 'Phone Number must be 10 char Long'
    elif len(customer.password) < 5:
        error_message = 'Password must be 5 char long'
    elif len(customer.email) < 5:
        error_message = 'Email must be 5 char long'
    elif customer.isExists():
        error_message = 'Email Address Already Registered..'
    # saving

    return error_message
```

cart.py

```
from django.db import models
from .product import Products
from .customer import Customer
import datetime

class Order(models.Model):
    product = models.ForeignKey(Products,
                                on_delete=models.CASCADE)
    customer = models.ForeignKey(Customer,
                                 on_delete=models.CASCADE)
    quantity = models.IntegerField(default=1)
    price = models.IntegerField()
    address = models.CharField(max_length=50, default='', blank=True)
    phone = models.CharField(max_length=50, default='', blank=True)
    date = models.DateField(default=datetime.datetime.today)
    status = models.BooleanField(default=False)

    def placeOrder(self):
        self.save()

    @staticmethod
    def get_orders_by_customer(customer_id):
        return Order.objects.filter(customer=customer_id).order_by('-date')
```

checkout.py

```
from django.shortcuts import render, redirect

from django.contrib.auth.hashers import check_password
from store.models.customer import Customer
from django.views import View

from store.models.product import Products
from store.models.orders import Order

class CheckOut(View):
    def post(self, request):
        address = request.POST.get('address')
        phone = request.POST.get('phone')
        customer = request.session.get('customer')
        cart = request.session.get('cart')
        products = Products.get_products_by_id(list(cart.keys()))
        print(address, phone, customer, cart, products)

        for product in products:
            print(cart.get(str(product.id)))
            order = Order(customer=Customer(id=customer),
                          product=product,
                          price=product.price,
                          address=address,
                          phone=phone,
                          quantity=cart.get(str(product.id)))
            order.save()
        request.session['cart'] = {}
        return redirect('cart')
```

orders.py

```
from django.shortcuts import render, redirect
from django.contrib.auth.hashers import check_password
from store.models.customer import Customer
from django.views import View
from store.models.product import Products
from store.models.orders import Order
from store.middlewares.auth import auth_middleware

class OrderView(View):

    def get(self, request):
        customer = request.session.get('customer')
        orders = Order.get_orders_by_customer(customer)
        print(orders)
        return render(request, 'orders.html', {'orders': orders})
```

Conclusion

The project already includes a lot of features. The main beneficiaries are both customers and administrators who take longer to behave online. In addition, additional features can be identified and incorporated in the future. It will take more time and effort to understand the need and adjust it to a computerized system to accommodate additional features.

References

1. Django Docs
2. About E-Commerce
3. HTML Docs
4. CSS Docs
5. Java Docs