Admin Controlled Web-Store Interface

Submitted in partial fulfilment of the requirements of the degree of

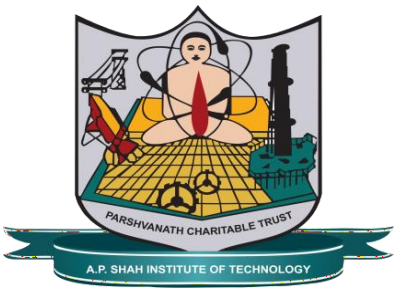
BACHELOR OF COMPUTER SCIENCE & ENGINEERING
ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING

by

Kaiser M. Momin,	21106009
Devesh M. Sali,	21106016
Ratnakar R. Pisal,	21105041
Yash P. Malvade,	21106032

Guide:-

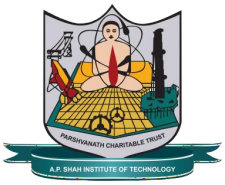
Prof. Mahesh Pawaskar



Department of Computer Science & Engineering
Artificial Intelligence & Machine Learning

A. P. SHAH INSTITUTE OF TECHNOLOGY

(2022 - 2023)

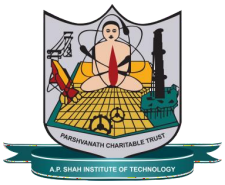


A. P. SHAH INSTITUTE OF TECHNOLOGY

CERTIFICATE

This is to certify that the Mini Project A entitled "**Admin Controlled Web-Store Interface**" is a bonafide work of **"Kaiser Momin (21106009), Devesh Sali (21106016), Ratnakar Pisal (21106041), Yash Malvade (21106032)"** submitted to the University of Mumbai in partial fulfilment of the requirement for the award of the degree of **Bachelor of Engineering** in **Computer Science & Engineering (Artificial Intelligence & Machine Learning).**

_____	_____
Project Guide	Head of Department
Prof. Mahesh Pawaskar	Prof. Jaya Gupta



A. P. SHAH INSTITUTE OF TECHNOLOGY

Project Report Approval for S.E.

This Mini project report entitled ***Admin Controlled Web-Store Interface*** by ***Kaiser Momin, Devesh Sali, Ratnakar Pisal, Yash Malvade*** is approved for the degree of ***Bachelor of Engineering*** in ***Computer Science & Engineering (Artificial Intelligence and Machine Learning) 2022 – 2023.***

Examiner Name Signature 1. _____

2. _____

Date:

Place:

Declaration

We declare that this written submission represents my ideas in my own words and where others' ideas or words have been included, We have adequately cited and referenced the original sources. We also declare that we have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/data/fact/source in my submission. We understand that any violation of the above will be cause for disciplinary action by the Institute and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been taken when needed.

_____

Kaiser M. Momin

_____

Devesh M. Sali

_____

Ratnakar R. Pisal

_____

Yash P. Malvade

Date:

Abstract

The main implication of the project is an Ecommerce website controlled by a single or multiple admins (superusers) created at the backend which allows a full authoritarian control of the database linked to the frontend. The database in question may be edited in real time and all edits/changes shall reflect directly on the website. Furthermore, the project focuses on providing the user a direct and seamless shopping experience. This is achieved using the Django web development framework which uses a codependent MTV (Media Template View) Architecture. The actual application of the project is to effectively manage an inventory hosted on a local or live server.

Also keeping in mind the ease of access for the Database and its management, the Product Inventory has been made accessible for editing via the Django Administration Portal. This ensures a safe and amenable database.

CONTENTS

Chapter 1: Introduction

When discussing the expanse of any E-commerce website we are often put up against the question of database management. This is the most crucial factor in the existence and proper functioning of any Ecommerce website or as we refer to it, a Webstore. In this project we face and deal with such issues on the basis of Inventory management which comes out to be the most important factor of any Webstore. Existing projects of the same kind include much more advanced systems such as Flipkart and Amazon.

1.1 Problem Definition:

Webstores must have the following interactive elements at the Front End:

i) Home Page: Open Product Catalogue

ii) Individual Categories: Sorting system for products as per product type

iii) Products in stated Categories: Placement of Products in correct Category

iv) Signin/Signup Pages: For Users to Interact with all elements and placing orders.

v) Cart module: To add multiple items and to purchase them at the same time.

v) Delivery Status: To follow through a Product's Delivery

And the Following editable content at the backend:

i) Admin Login: To prevent all unauthorised ingress.

ii) Editable Inventory: To add/update products from time to time.

iii) Inventory history: To store all edits made in the inventory.

iv) Admin Creation: To add more admins for large scale addition of products.

v) Real Time Product Updation: To enable quick changes and reflection on website.

1.2 Objectives:

i) Primary Objective of the project is to create a Webstore with a real time editable and quick updating backend which enables a clean and concise workflow environment.

ii) Secondary Objective of the project is to provide a smooth Website Frontend following traditional Ecommerce Website norms.

iii) Also to elevate both User and Admin Experience in Webstore Usage.

1.3 Scope

The Project aims to use the Django MTV Architecture to realise the easy link between Frontend and Database Editing. The limitation provided by the framework is one of speed and learning curve. Framework similar to Django is Ruby on Rails which is about 7% faster making it more efficient. But due to its smoother learning curve, Django is equally preferred amongst full stack developers. Using the Common Media-Template-View architecture the need of relinking of separate modules is eliminated. This also brings into highlight the main function of the project, which is to provide an easy to use and fast reflecting Inventory. It is also used due to its popularity in having a DRY (Don't Repeat Yourself) approach.

Chapter 2: Literature Survey

2.1 Amazon:

Amazon is one of the most popular Ecommerce websites in existence.

For the Frontend it uses:

i) JavaScript: For User Interface (UI) Design.

For the Backend it uses:

i) Java

ii) C++

iii) Perl

For the Database it uses:

i) DynamoDB

ii) RDS/Aurora

iii) RedShift

As it is seen that JavaScript (Hereon referred to as JS) is one of the most popular Scripting Languages, it is used for its vast applications amongst many Ecommerce stores.

Amazon is the most popular Ecommerce Website in the World. But in order to make the Inventory management easy and fast Amazon creates Sub-Inventories under the name of each and every seller that has a product to sell. However this causes another issue to arise, when more than one seller exists, this is the creation of more than one repetition of the same product as the listing as multiple sellers sell the same exact product.

This is avoided by not allowing multiple sub-admins within the original framework of the Website and hence no repetitive occurrence of items is seen.

2.2 Etsy:

For the Frontend it uses:

i)JavaScript

For the Backend it uses:

i) PHP

For the Database it uses:

i)MySQL

ii) Redis

JavaScript (JS) is extremely popular in the ecommerce world because it helps create a seamless and user-friendly experience for shoppers. Take, for instance, loading items on category pages, or dynamically updating products on the site using JS. The Database is one of the simplest to exist using both MySQL and Redis which are both open source and are simple to learn.

Etsy uses PHP for its backend and hence most troubles seen in Amazon due to usage of multiple languages for the backend are resolved using a singular language for backend management. Also the usage of Open Source languages for Database management makes it easier to find a larger and more versatile developer base for the database and all related functions.

However the use of Open Source software to manage a website poses a threat of personal security of the users and the website as well. The usage of a more security focused Database management framework such as Django gives peace of mind to both the user and the Developer.

Chapter 3: Technology Stack.

The Admin Controlled Webstore Interface uses the following pattern to implement the complete technology stack of the Project.

3.1 Frontend:

i)HTML: The simplest and safest option in Frontend design it is the most easy to use programming language for the frontend.

3.2 Backend:

i)Python:

Chosen due to the Following Reasons:

a)Python is relatively easy to learn.

b) Mature frameworks and development tools.

c)Clean syntax and easy-to-read code.

d)Python is a universal language.

e)A large active community around Python.

SubSection Explaining The Django Framework:

Django is a popular Web development framework which allows the user to complete full stack development using only Django at the backend. This enables the

3.3 Database:

i)MySQL:

Chosen due to the Following Reasons:

a)Open-source and compatible: This simply means that anyone can install and use the basic software, while also enabling third parties to modify and customise the source code.

b)Fast and reliable: MySQL was developed for speed, even if this may come at the expense of some additional features

c)Availability: Online businesses and web platforms need to be able to provide round-the-clock services for a global audience. This is why high availability is a core feature of MySQL.

d)Scalability: As data volumes and user loads increase, the database store needs to be scaled-up. It must be able to cope with the additional workload without a drop in performance.

e)Security: This is always an important consideration for businesses as they need to protect sensitive data and defend against cyberattacks. MySQL offers encryption using the Secure Sockets Layer (SSL) protocol, data masking, authentication plugins, and other layers of security to protect data integrity.

Chapter 4: Benefits and Applications

4.1 Economic Benefits of Ecommerce:

E-commerce is expanding quickly and is expected to reach $1 trillion in sales by 2022. The current ecommerce environment is stimulating entrepreneurship and encouraging firms of all sizes to compete because it has a larger market share and lower entry hurdles than traditional brick-and-mortar commerce. All clients will have access to more online experiences thanks to this steady development and activity.

Because it provides customers with convenience, value, and choice, ecommerce has become more and more popular. As more people shop online, the new economy grows and sellers are inspired to innovate, identifying what makes their product special and developing a memorable customer experience. Customers are more motivated to shop online as a result, which boosts economic growth in general.

4.2 Environmental Benefits of Ecommerce:

Beyond the ways in which e-commerce can boost a nation's economy, it can also lessen its carbon footprint. Global research by Generation IM found that e-commerce is 17% more carbon-efficient than conventional retail outlets. The business's carbon-efficient operations cover areas like product transportation, construction, warehousing, product packaging, and manufacturing. Online businesses have greater flexibility when it comes to putting policies in place that can lower their carbon footprints because they depend less on having physical store locations.

Consumers today choose sustainable companies more and more because they can teach them how to make better purchasing decisions. Leading businesses are responding to this consumer demand by leveraging their clout to embrace more environmentally friendly practices and increase customer trust. As ecommerce brands continue to make real and impactful changes to promote sustainability, this will lead to more positive long-term benefits for the environment.

4.3 Benefits of Ecommerce to Society:

With an e-commerce business, you'll reach a global audience instead of a specific location and increase profits. To society, e-commerce has increased online education. Students can study virtually and also work part-time. It reduces traffic and air pollution since people don't go to stores physically they shop online.

4.4 Applications:

1. Retail

E-retailing, often known as online retailing, is the sale of products and services by businesses to customers via online stores. This is done through the use of tools such as virtual shopping carts and e-catalogs. There are several e-commerce applications in this industry.

2. Accounting

Finance and e-commerce are more intertwined than ever before. Banks and stock exchanges make extensive use of e-commerce in their operations. Balance checks, bill payments, money transfers, and more services are available through online banking. Online stock trading allows users to trade stocks online by providing information about equities such as performance reports, analysis, charts, and so on via websites.

3. Production

In the manufacturing industry, e-commerce serves as a platform for firms to conduct electronic transactions. Groups of firms can carry out their activities more smoothly by combining purchasing and selling, exchanging market conditions, inventory check information, etc.

4.4 Applications(Continued):

4. Trade

Applying e-commerce to trade elevates it to a higher level, allowing individuals to participate without regard for geographical borders. This encourages more participation, more bargaining and contributes to the success of the trade.

5. Advertising

Development and commercialization strategies like pricing, product characterization, and customer relationship can be boosted by utilising e-commerce. This will give consumers a more enriched and personalised purchasing experience. Digital marketing tactics have grown in importance as a means of promoting enterprises.

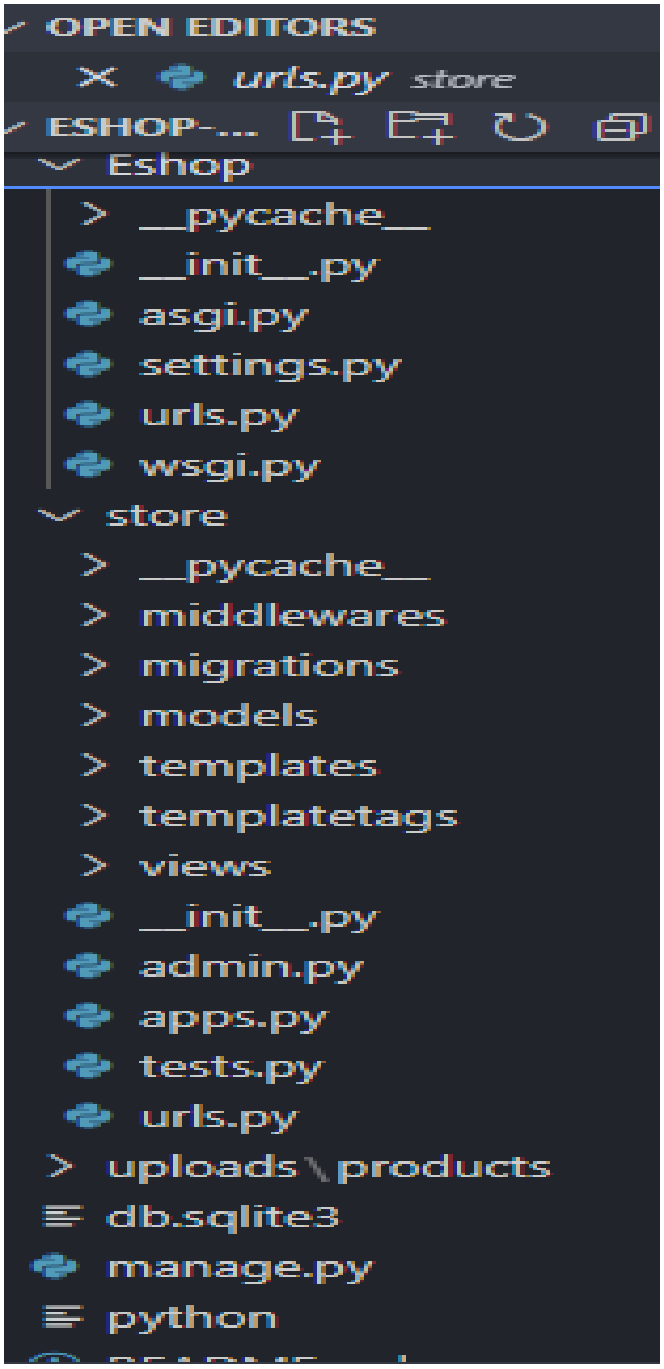
6. Digital Shopping

People's buying habits have shifted dramatically in the previous several years. "Go online" has become a success mantra for all enterprises. Online shopping is easy, pleasant, and, in most cases, inexpensive. The success of online shopping applications like Flipkart and Amazon demonstrates this.

Chapter 5: Project Design

Django is a powerful framework based on python. This project includes storing products in the database and showing them on the website.

5.1 Directory Structure:



5.2 Django MTV Architecture:

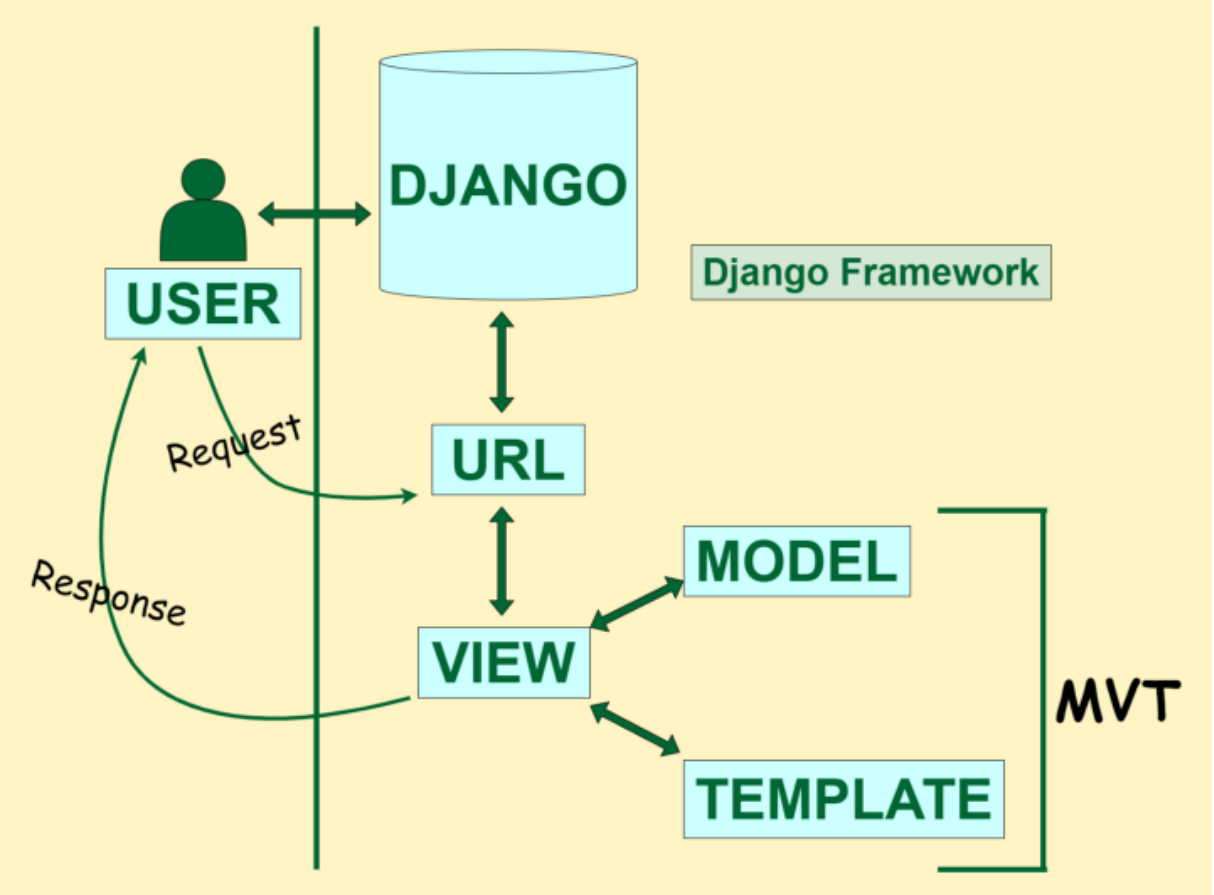
MVT stands for Model-View-Template. Sometimes it is also referred to as MTV(Model-Template-View). MVT is a design pattern or design architecture that Django follows to develop web applications. It is slightly different from the commonly known MVC(Model-View-Controller) design pattern.

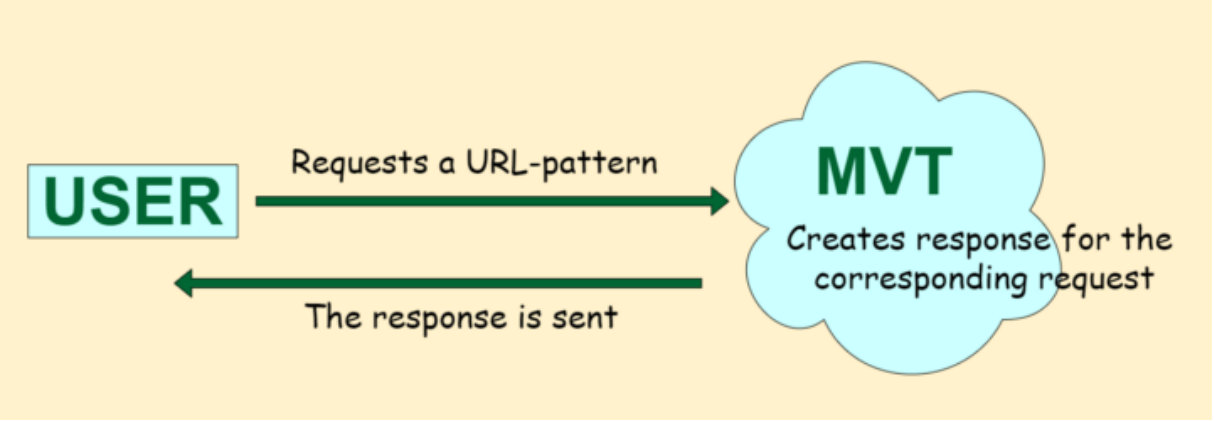
MVT determines the total structure and workflow of a Django application. In an MVT architecture —

- The Model manages the data and is represented by a database. A model is basically a database table.
- The View receives HTTP requests and sends HTTP responses. A view interacts with a model and template to complete a response.
- The Template is basically the front-end layer and the dynamic HTML component of a Django application.

5.2 Django MTV Architecture:

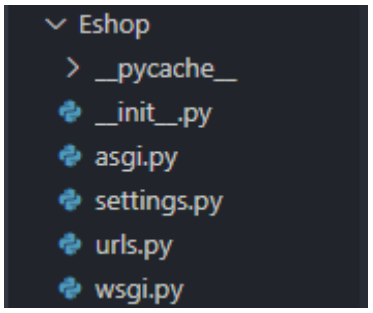
(Diagrammatic Explanation)





5.3 Modules of system:

5.3.1 E-Shop



i) __init__.py

Python defines two types of packages, regular packages and namespace packages. Regular packages are traditional packages as they existed in Python 3.2 and earlier. A regular package is typically implemented as a directory containing an __init__.py file. When a regular package is imported, this __init__.py file is implicitly executed, and the objects it defines are bound to names in the package's namespace. The __init__.py file can contain the same Python code that any other module can contain, and Python will add some additional attributes to the module when it is imported.

ii) settings.py

The settings.py is the central configuration for all Django projects. In previous chapters you already worked with a series of variables in this file to configure things like Django applications, databases, templates and middleware, among other things.

iii) urls.py

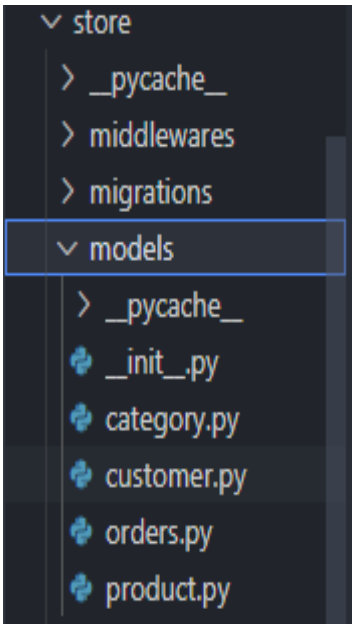
This tells Django to search for URL patterns in the file books/urls.py . For example, A URL request to /books/crime/ will match with the second URL pattern. As a result, Django will call the function views.

iv)wsgi.py

The main use of deploying with WSGI is the application callable which the application server uses to communicate with your code. It's commonly provided as an object named application in a Python module accessible to the server.

5.3.2 Store

5.3.2.1: Models



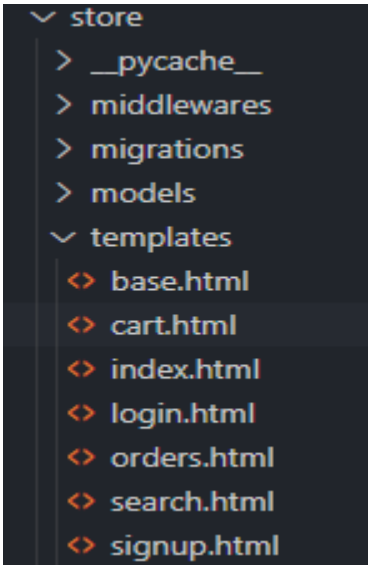
i) category.py: Manages all Django Framework relating to the Category class

ii) customer.py: Manages all customer data stored at the backend.

iii) orders.py: Manages all order related data stored at the backend, this is linked to the customer.py

iv) product.py: Manages the product catalogue , this is linked to the category.py module and hence makes the sorting of products according to the database Categories.

5.3.2.2: Templates



i)base.html.

ii)cart.html

iii)index.html

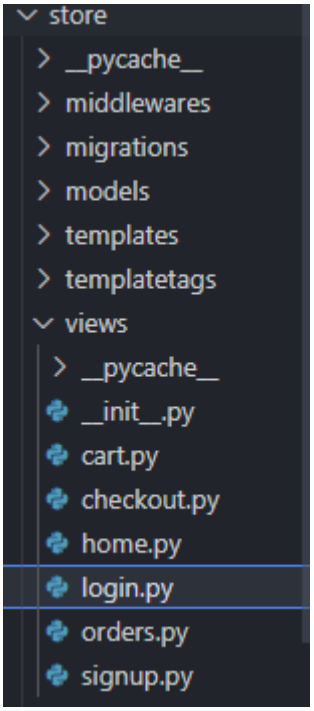
iv)login.html

v)orders.html

vi)search.html

vii)signup.html

5.3.2.3: Views



i) cart.py: Gives the Cart.html page all of its functionality.

ii) checkout.py: Gives the checkout.html page all of its functionality.

iii) home.py: Gives the base.html page, all of its functionality.

iv)orders.py: Gives the orders.html page all of its functionality.

v) signup.py: Gives the signup.html page all of its functionality.

5.3.2.4: Main Modules

i) admin.py: this module allows access to the Django Administration Panel, and hence it creates the entire editable module. It also allows access to the creation of superusers and editing their authority.

ii) apps.py: this module initialises Django Framework within the program for Database Linking with the DB SQLite Server.

iii) test.py: this module creates the complete bug testing application for the entire website.

iv)url.py: this module defines all the links between multiple pages within the project.

Chapter 6: Project Implementation

6.1 Modules for controlling HTML pages:

6.1.1 manage.py module:

manage.py > ...

```python
#!/usr/bin/env python
"""Django's command-line utility for administrative tasks."""
import os
import sys


def main():
    """Run administrative tasks."""
    os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'Eshop.settings')
    try:
        from django.core.management import execute_from_command_line
    except ImportError as exc:
        raise ImportError(
            "Couldn't import Django. Are you sure it's installed and "
            "available on your PYTHONPATH environment variable? Did you "
            "forget to activate a virtual environment?"
        ) from exc
    execute_from_command_line(sys.argv)


if __name__ == '__main__':
    main()
```

6.1.2 cart.py module:

cart.py

store > views > cart.py > ...

```python
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

class Cart(View):
    def get(self , request):
        ids = list(request.session.get('cart').keys())
        products = Products.get_products_by_id(ids)
        print(products)
        return render(request , 'cart.html' , {'products' : products} )

```

6.1.3 home.py module:

```
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)
```

6.1.4 orders.py module:

orders.py

store > views > orders.py > ...

```
1   from django.shortcuts import render, redirect
2   from django.contrib.auth.hashers import check_password
3   from store.models.customer import Customer
4   from django.views import View
5   from store.models.product import Products
6   from store.models.orders import Order
7   from store.middlewares.auth import auth_middleware
8
9   class OrderView(View):
10
11
12      def get(self , request ):
13          customer = request.session.get('customer')
14          orders = Order.get_orders_by_customer(customer)
15          print(orders)
16          return render(request , 'orders.html'  , {'orders' : orders})
17
```

6.1.5 signup.py module:

```
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'
```

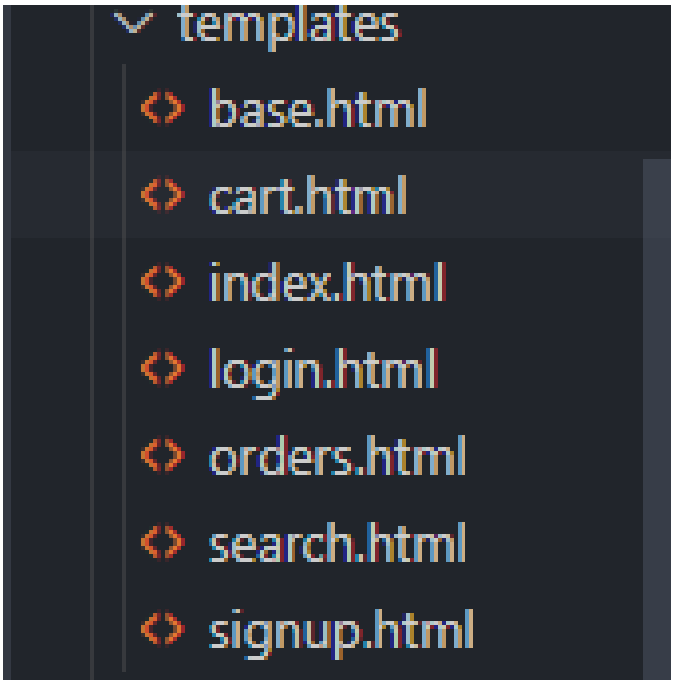
```
        elif len (customer.last_name) < 3:
            error_message = 'Last Name must be 3 char long or more'
        elif not customer.phone:
            error_message = 'Enter your Phone Number'
        elif len (customer.phone) < 10:
            error_message = 'Phone Number must be 10 char Long'
        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
```

6.1.6: login.py module:

```
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')
```

6.2 HTML Pages created for the frontend:



6.3 admin.py (for Database Control):

admin.py

store > admin.py > ...

```python
from django.contrib import admin
from .models.product import Products
from .models.category import Category
from .models.customer import Customer
from .models.orders import Order


class AdminProduct(admin.ModelAdmin):
    list_display = ['name', 'price', 'category']


class CategoryAdmin(admin.ModelAdmin):
    list_display = ['name']

# Register your models here.
admin.site.register(Products,AdminProduct)
admin.site.register(Category)
admin.site.register(Customer)
admin.site.register(Order)


```

6.4 urls.py (for managing links between the pages):

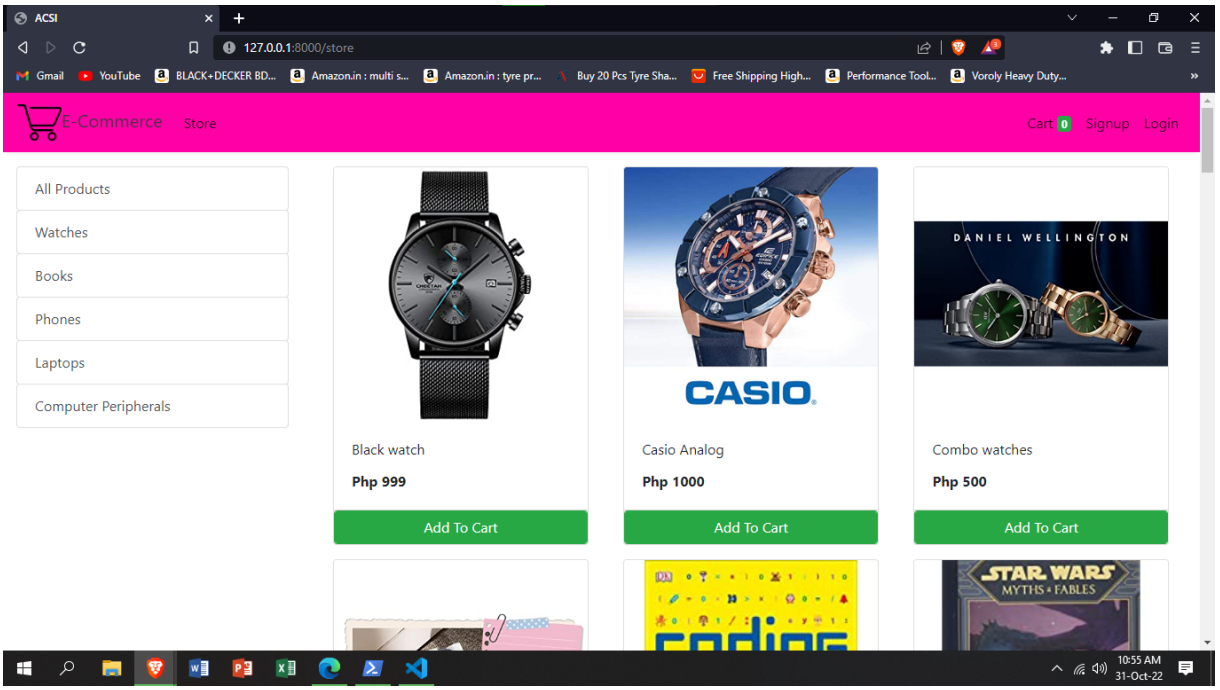
urls.py

store > urls.py > ...

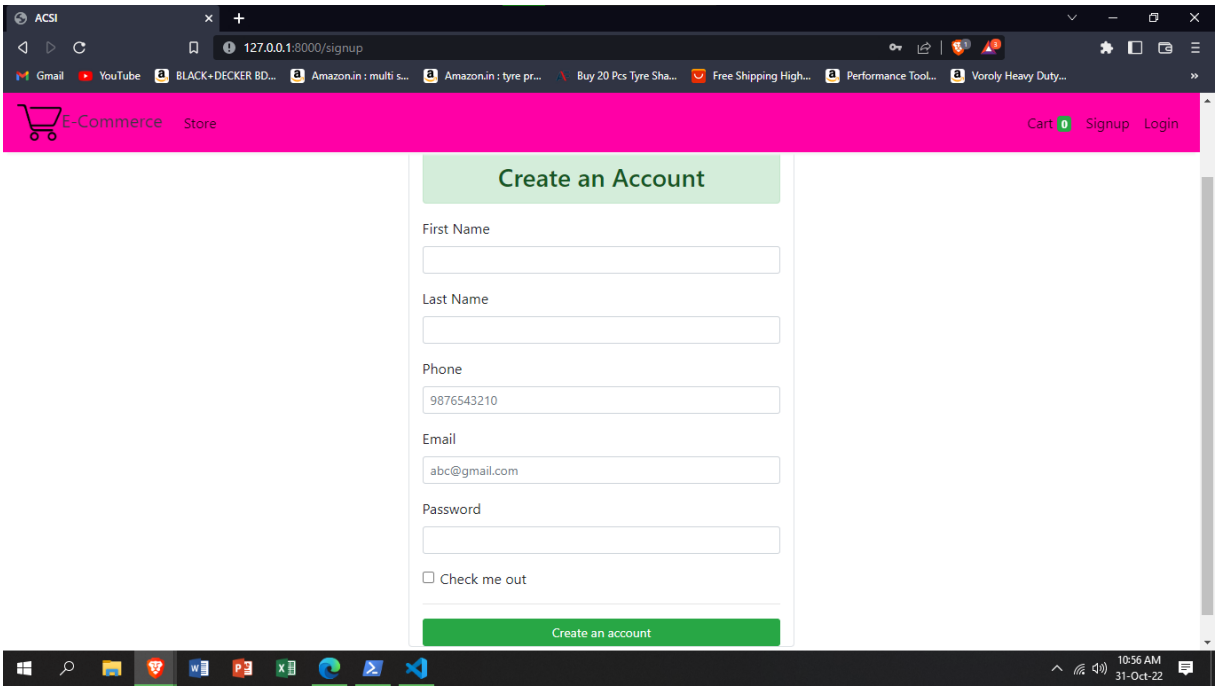
```python
from django.contrib import admin
from django.urls import path
from .views.home import Index , store
from .views.signup import Signup
from .views.login import Login , logout
from .views.cart import Cart
from .views.checkout import CheckOut
from .views.orders import OrderView
from .middlewares.auth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

]
```

Chapter 7: Result

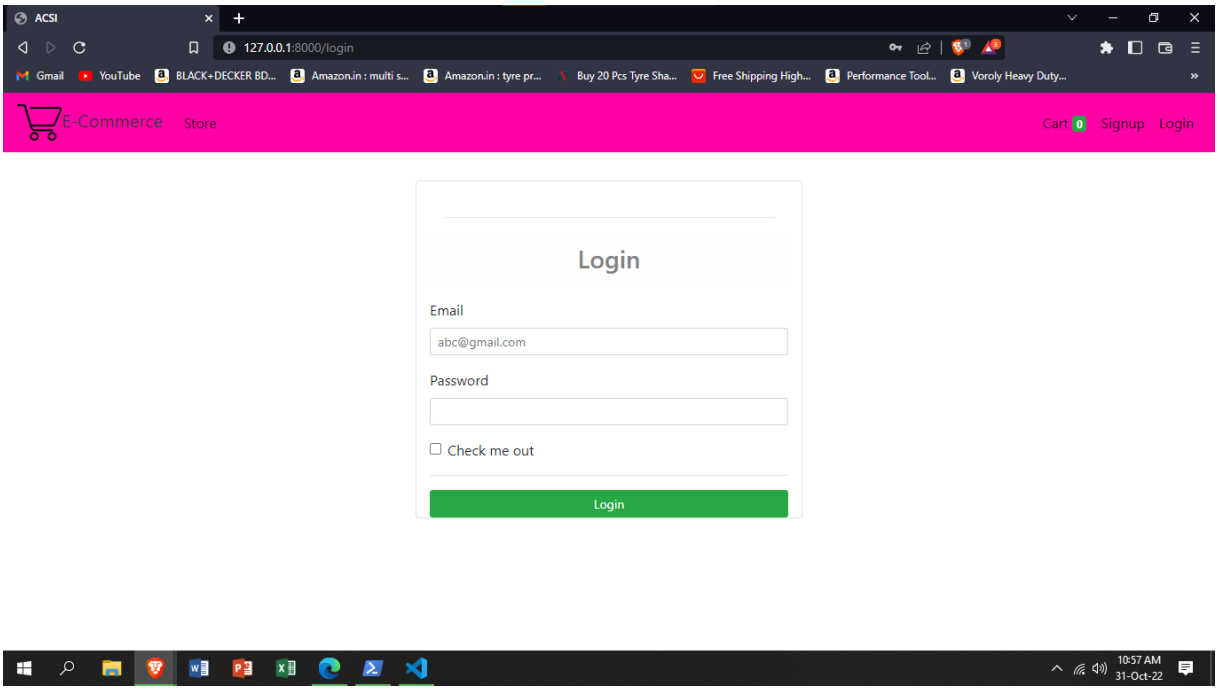
7.1 Homepage:



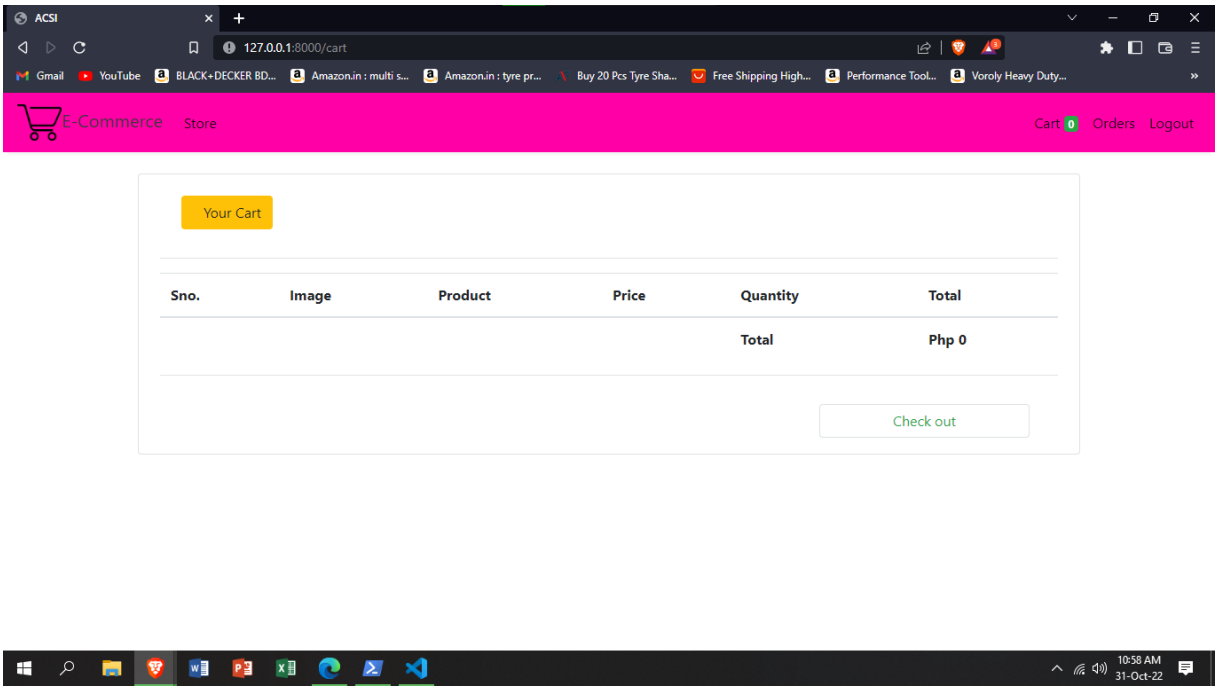
7.2 Signup Page:



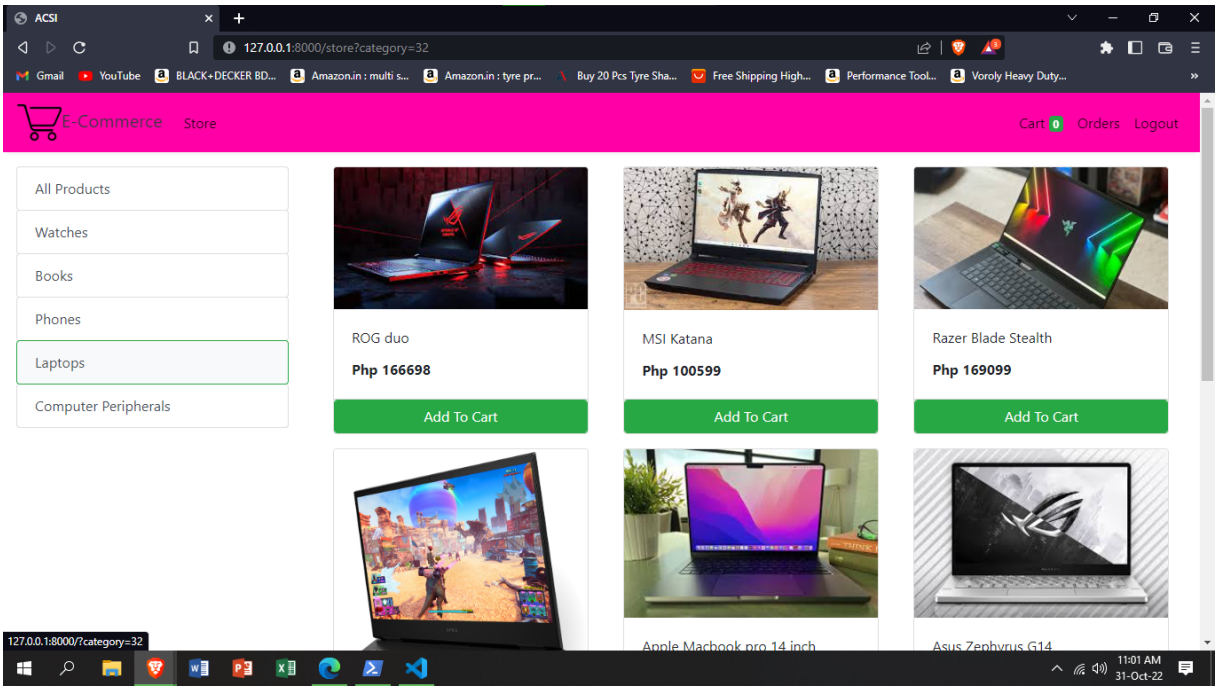
7.3 Login Page:



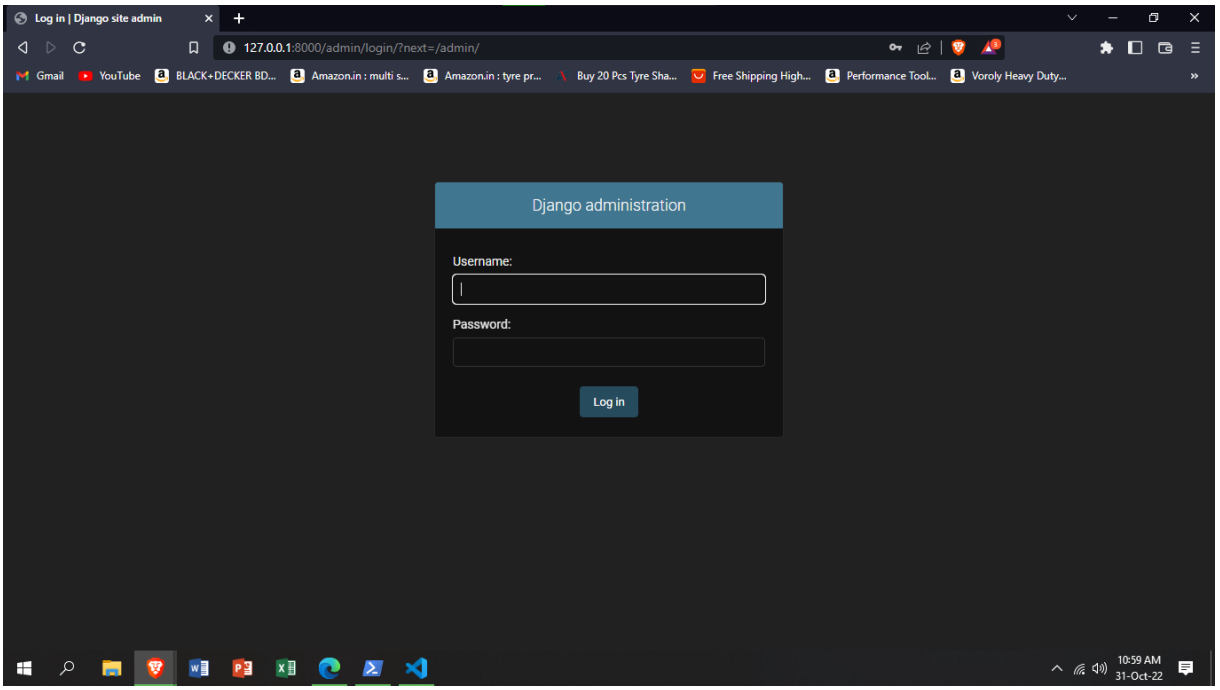
7.4 Cart Page:



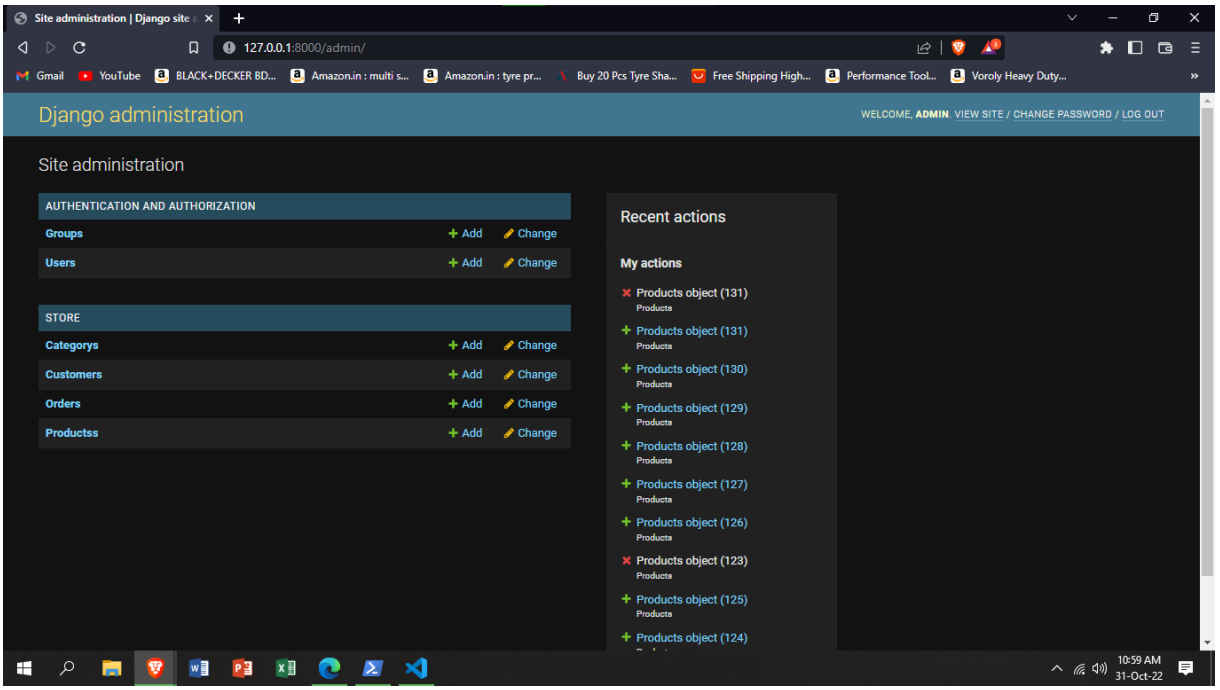
7.5 Specific Selected Category Page:



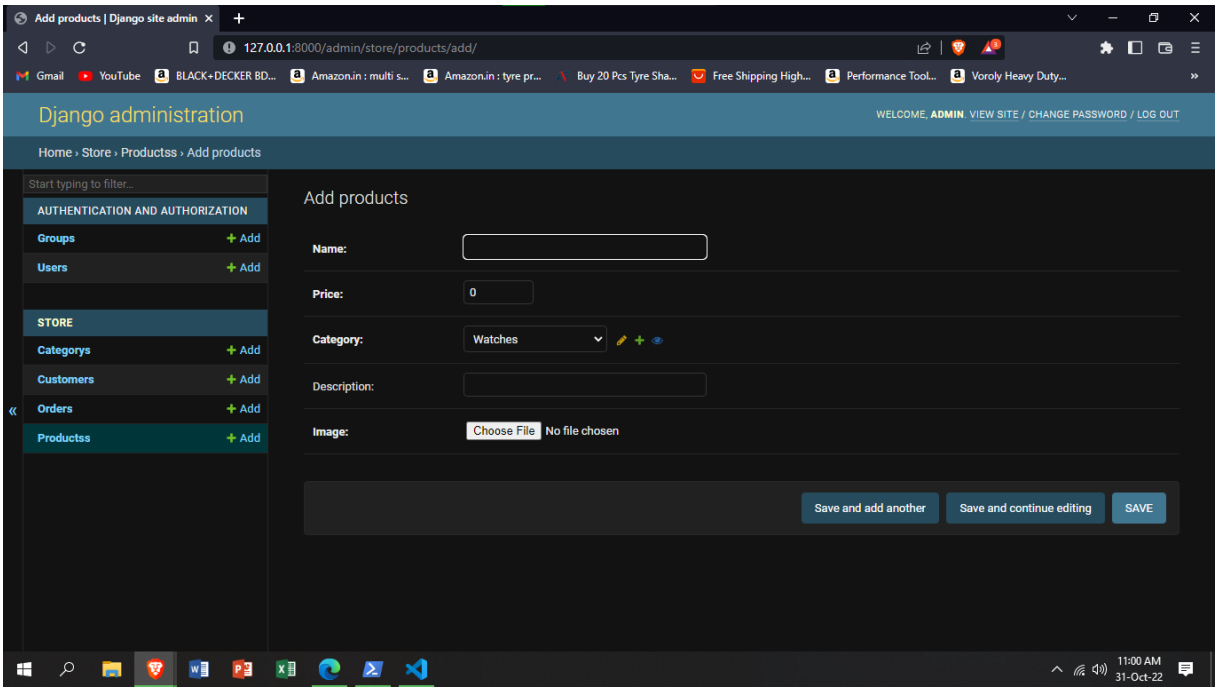
7.6 Django Admin Login:



7.7 Django Admin Page:



7.8 Product Addition Page:



Chapter 8: Conclusion

In conclusion we have achieved the following goals:

1) A completely functional Webstore interface was designed and implemented, this Interface has an admin controlled website inventory which makes the backend and the database entirely secure and non-editable without the admin access.

2) The Inventory is directly linked to the SQL database via the Django Framework and since the entire project is Created in Visual Studio Code, it uses the DB SQLite extension to maintain a link between the live backend and the project database.

3) The designed Webstore has a real time editable and updating inventory which enables the programmer to verify necessary updates and make changes immediately if necessary.

4) The Simplicity of the Django framework allows the programmer to adapt to the MTV architecture easily and effectively. Also the readability of Python code is simple and fast due to which all necessary edits needed are quick and easy.

References:

Software:

i) https://www.python.org/downloads/ (for python 3.11.0)

ii) https://code.visualstudio.com/docs/?dv=win (for VS code)

iii) https://www.sqlite.org/download.html (for independent SQLite usage)

iv) Extensions in VS code:

a) DB SQLite by alexcvzz
b) Atom One Dark Theme by Mahmoud Ali

Websites:

i) https://python.plainenglish.io/the-mvt-design-pattern-of-django-8fd47c61f582

ii) https://www.geeksforgeeks.org/top-10-reasons-to-choose-django-framework-for-your-project/

iii)https://uvik.net/blog/django-vs-ruby-on-rails/#:~:text=If%20we%20compare%20the%20two,Flexibility.

iv)https://www.jobsity.com/blog/5-reasons-why-mysql-is-still-the-go-to-database-management-system

v)https://www.geeksforgeeks.org/django-project-creating-a-basic-e-commerce-website-for-displaying-products/

Videos:

i) https://www.youtube.com/watch?v=rHux0gMZ3Eg

ii) https://www.youtube.com/watch?v=YZvRrldjf1Y

iii) https://www.youtube.com/watch?v=_uQrJ0TkZlc

Appendix:

i)Django: Django is a free and open-source, Python-based web framework that follows the model–template–views architectural pattern. It is maintained by the Django Software Foundation.

ii)SQL: SQL is a domain-specific language used in programming and designed for managing data held in a relational database management system, or for stream processing in a relational data stream management system

iii) VS Code: Visual Studio Code, also commonly referred to as VS Code, is a source-code editor made by Microsoft with the Electron Framework, for Windows, Linux and macOS. Features include support for debugging, syntax highlighting, intelligent code completion, snippets, code refactoring, and embedded Git.

iv) pip: pip is the de facto and recommended package-management system written in Python and is used to install and manage software packages. It connects to an online repository of public packages, called the Python Package Index

v) MTV Architecture: Sometimes it is also referred to as MTV(Model-Template-View). MVT is a design pattern or design architecture that Django follows to develop web applications. It is slightly different from the commonly known MVC(Model-View-Controller) design pattern.

vi) URLs: A Uniform Resource Locator, colloquially termed a web address, is a reference to a web resource that specifies its location on a computer network and a mechanism for retrieving it. A URL is a specific type of Uniform Resource Identifier, although many people use the two terms interchangeably.