PROJECT REPORT:

TRENDYKRAFT (ECOMMERCE) - YOUR ULTIMATE FASHION DESTINATION

By

NITIKA GOYAL

ID:MST01-0029

Abstract

The Django E-commerce Trendy Kraft project is a sophisticated online shopping platform designed to offer users a seamless and intuitive experience. Built on Django framework, it boasts robust features including efficient product management, secure payment gateways, and responsive design. With a focus on scalability and flexibility, the platform aims to revolutionize the online shopping landscape, empowering businesses to thrive in the digital marketplace.

In summary, the Django E-commerce project represents a culmination of cutting-edge technology, meticulous development, and user-centric design, poised to revolutionize the online shopping landscape and empower businesses to thrive in the digital marketplace.

Table of Contents:

1. Introduction:

The TrendyKraft project is a ecommerce website developed using Django framework. It incorporates various sections such as Login Page, Dashboard, Store Page, Cart Page, Payment Page, and Invoice Page to provide users with a comprehensive shopping cart experience.

With Django's built-in features like ORM (Object-Relational Mapping), authentication, and security measures, coupled with its extensive ecosystem of third-party libraries and plugins, developing a feature-rich online store becomes both efficient and enjoyable.

From managing product catalogs and inventory to handling payments securely and providing a seamless shopping cart experience, Django empowers developers to craft intuitive, highperformance e-commerce websites that captivate customers and drive conversions.

2. Project Overview:

The project integrates Django framework along with Bootstrap for frontend styling and responsiveness. It features an enhanced admin interface using Django inbuilt Admin for easy customization. The website support and utilizes Decouple for security. Key functionalities include products display, email-activation, store page, adding and removing items from cart, and payment invoice and PayPal integration. Internationalization and localization support are provided using Django's built-in internationalization packages and middleware.

3. Technologies Used:

Django 5.0.1

Bootstrap 4

Python Decouple

Gmail (for smtp config)

SQLite3 (default database engine)

Django Middleware

H.T.M.L

C.S.S

JavaScript

JQuery

Ajax

PayPal Integration

Django 5.0.1: Django is a high-level Python web framework that facilitates rapid development of secure and maintainable websites and web applications. It follows the model-view-template (MVT) architectural pattern.

Bootstrap 4: Bootstrap is a popular front-end framework for building responsive and mobile-first web projects. It provides pre-designed templates and components using HTML, CSS, and JavaScript for creating user interfaces.

Python Decouple: It helps in strict separation of settings from code. helps you to organize your settings so that you can change parameters without having to redeploy your app.

CSS: CSS (Cascading Style Sheets) is a stylesheet language used to style the presentation and layout of HTML documents. It defines the styles, such as colors, fonts, margins, and padding, to enhance the visual appearance of web pages.

JavaScript: JavaScript is a high-level programming interactivity, dynamic behavior, and functionality to developers to manipulate the HTML and CSS of a interactions, and perform asynchronous operations.

HTML: It stands for Hyper Text Markup Language. It is the standard markup language for creating Web pages. It describes the structure of a Web page. It consists of a series of elements. Its elements tell the browser how to display the content.

Jquery: It makes Javascript easy to use on website.

4. System Architecture:

The system architecture follows a typical Django MVC (Model-View-Controller) pattern. It involves models for database management, views for handling user requests and rendering templates, and templates for HTML rendering. Middleware such as 'django.middleware.security.SecurityMiddleware' and 'django.middleware.locale.LocaleMiddleware' are utilized for security and localization respectively.

5. Implementation Details:

Home Page: Displays Trendy Kraft page where products are listed using HTML, CSS and JavaScript and Django. Provides user-friendly interface for searching products by category. Internationalization and localization are implemented using Django's built-in internationalization packages and middleware such as 'django.middleware.locale.LocaleMiddleware'.

Login Page: The login page serves as the entry point for users to access their accounts. It typically includes fields for users to input their username/email and password.

Store Page (Product Display): The store page is where users can browse through all the products available for purchase. Each product is displayed with relevant information such as name, price, description, and possibly images. I have categorized products to enhance the user experience. Implementing pagination can also help manage large product catalogs efficiently.

Cart: The cart is where users can view the items they have added for purchase. It display a summary of each item, including its name, quantity, price, and total cost. Users are able to adjust quantities, remove items, and possibly apply discounts or promotional codes. Ensure that the cart state is persistent across user sessions, possibly by storing it in the session or associating it with the user's account.

Payment (PayPal Integration): The payment page is where users initiate the checkout process and complete their purchase. Users are redirected to PayPal's secure payment gateway to authorize the transaction. After the payment is processed successfully, users are redirected back to site, where they can update the order status and generate an invoice. They can also provide clear feedback after purchase .

Invoice Generation Page: The invoice generation page is where users can view and download invoices for their purchases. Each invoice includes details such as the order number, date, items purchased, quantities, prices, taxes, and total amount due.

Dashboard: A separate page for dashboard is provided to user where he/she can update their address, profile picture, change password view their previous orders and view invoice of orders.

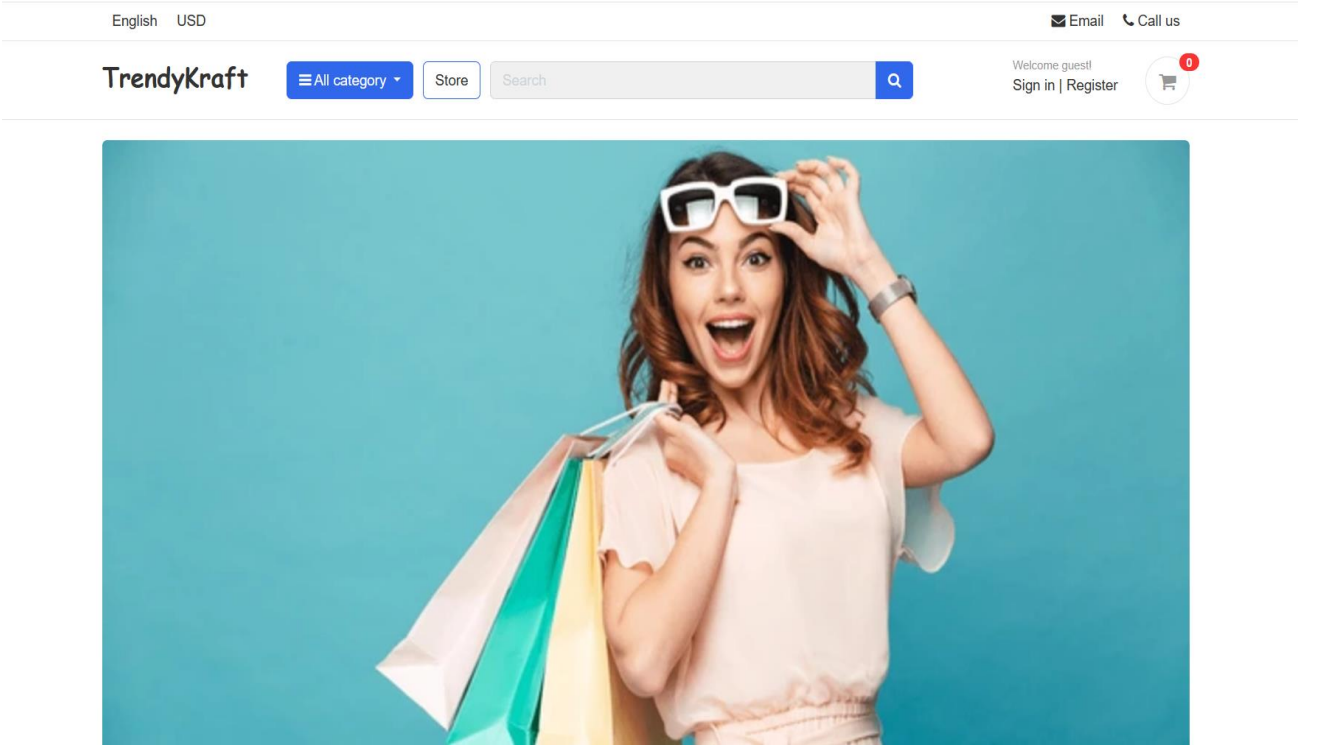
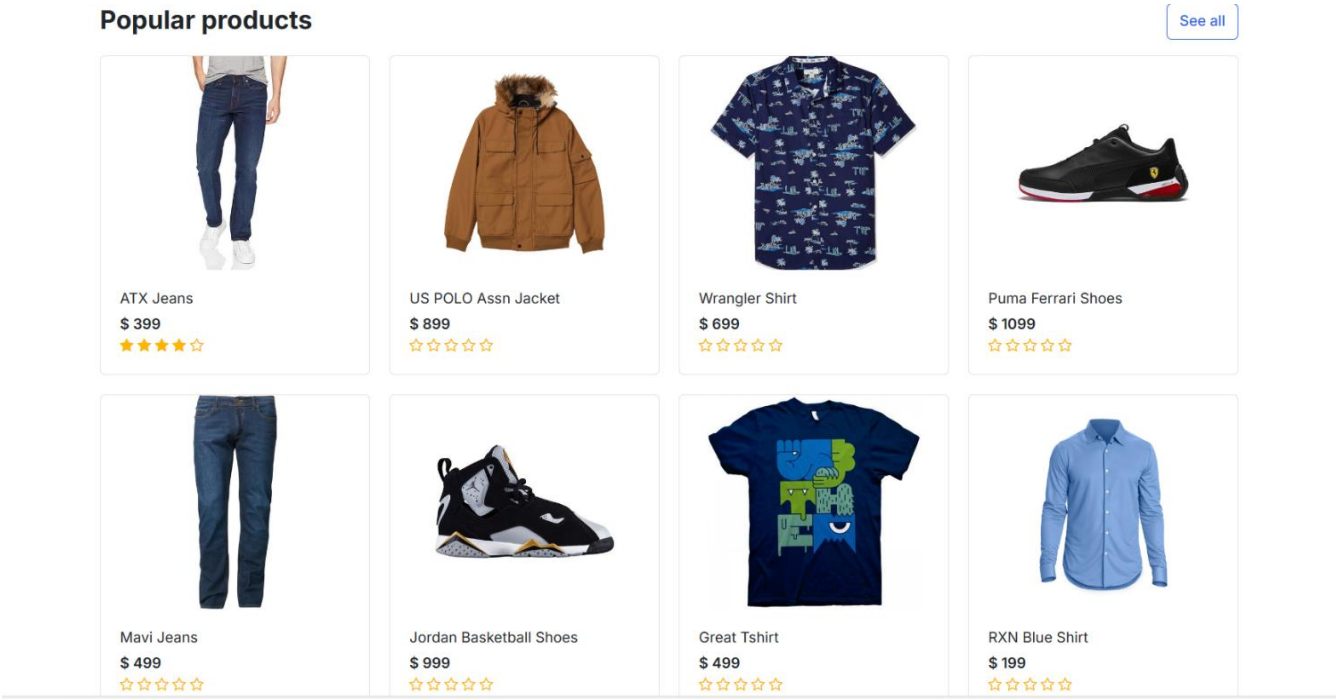


Fig 1: Home page



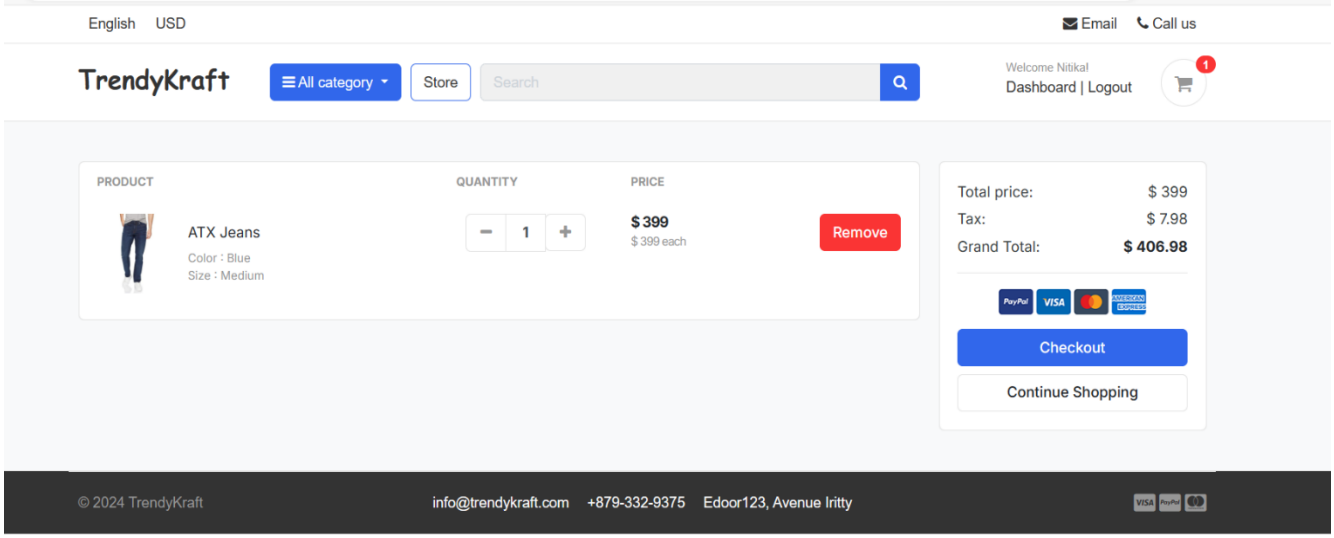
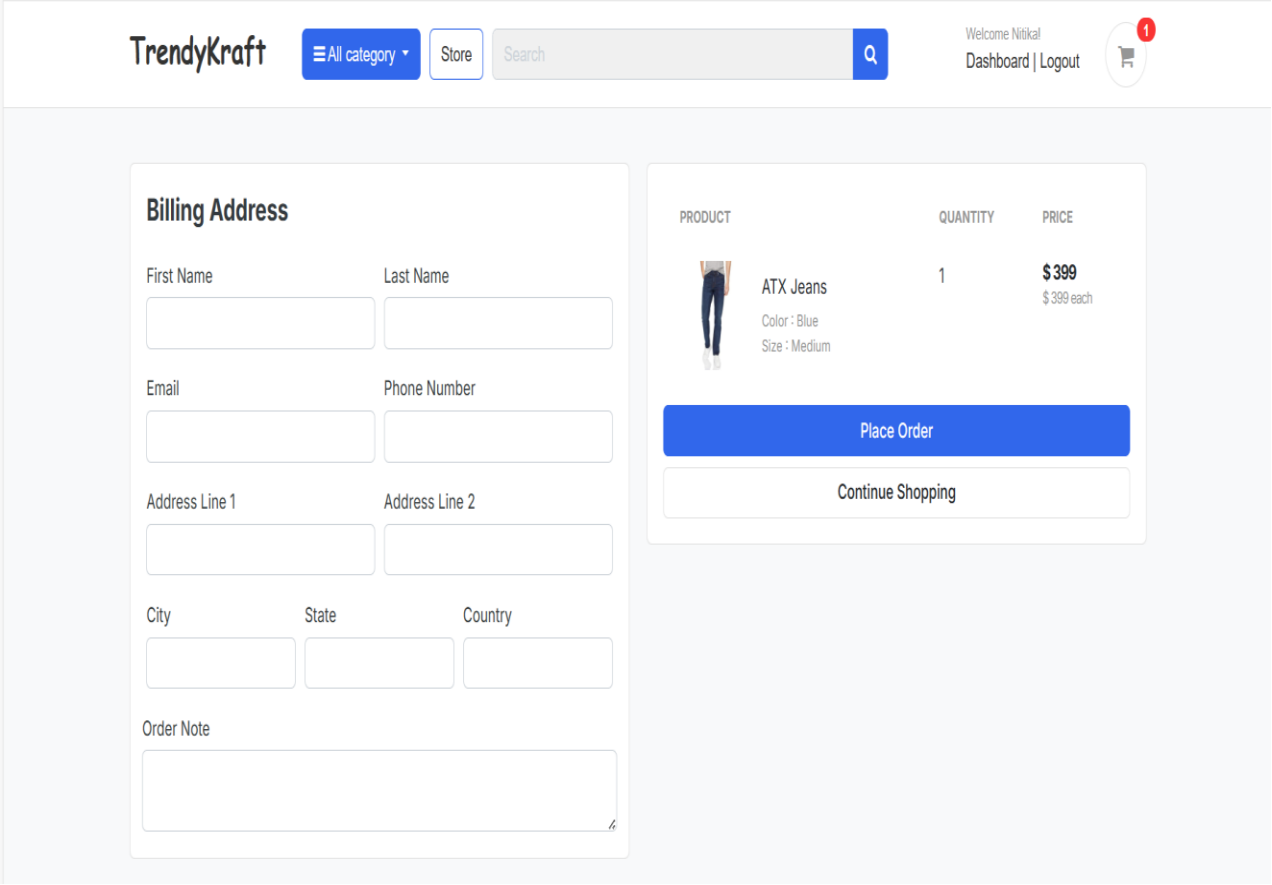


Fig2.Cart page

Fig 3. Address Page



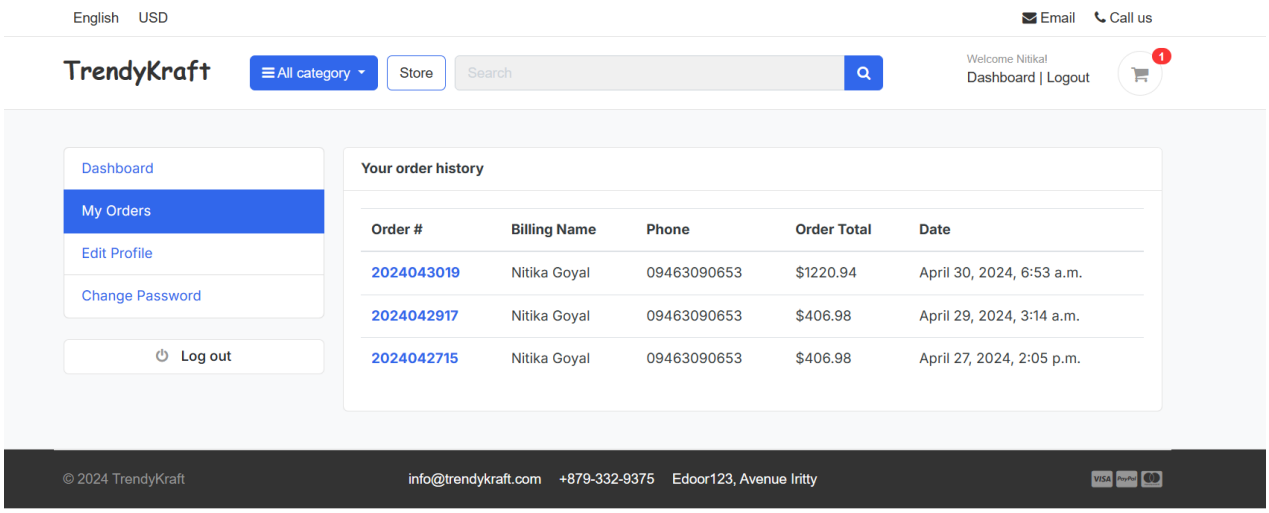
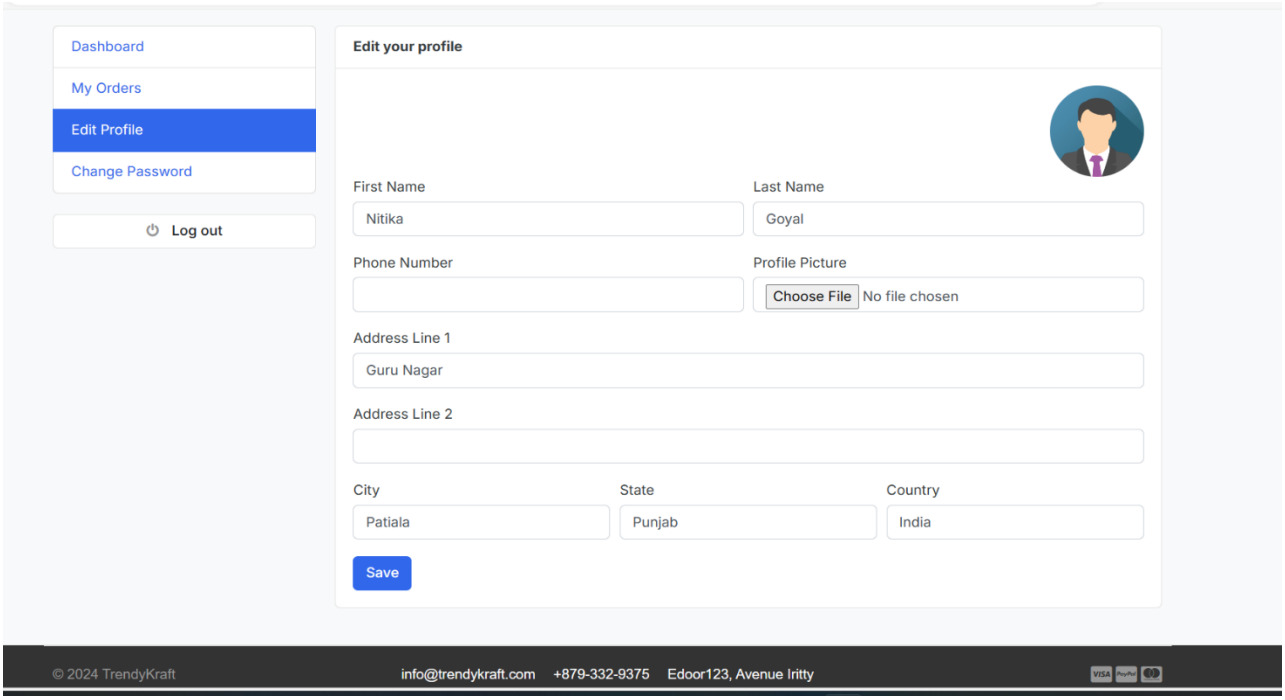


Fig4.Dashboard Features



GreatKart

Invoiced To
Nitika Goyal
Guru Nagar
Patiala, Punjab
India

Order #2024043019
Transaction ID 40E94858HN3836647
Order Date: April 30, 2024, 6:53 a.m.
Status: COMPLETED

Products	Qty	Total
Great Tshirt Color : Only Color Size : Free Size	1	$499.0 USD
Mavi Jeans Color : Only Color	1	$499.0 USD
RXN Blue Shirt Color : Blue Size : Free Size	1	$199.0 USD
	Sub Total:	**$1197.0 USD**
	Tax:	**$23.94 USD**
	Grand Total:	**$1220.94 USD**

Thank you for shopping with us!

Fig7. Invoice

Fig8. Review Page

Write Your Review

How do you rate this product?

★★★★★

Review Title:

Review:

Submit Review

Customer Reviews

★★★★☆ 1 reviews

Nitika Goyal April 29, 2024, 4:16 a.m.

★★★★☆

Pants quality

quality was awesome

Home Page:

Functionality:

Displays products in store category, search bar, sign in and register using HTML, CSS, JavaScript and Django.

Provides a user-friendly interface for searching products by name.

Localization support using Django's built-in packages and middleware.

Technologies Used:

Frontend: HTML, CSS, JavaScript

Backend: Django

Middleware: 'django.middleware.locale.LocaleMiddleware'

Views.py

```
from django.shortcuts import render
from store.models import Product, ReviewRating

def home(request):
    products = Product.objects.all().filter(is_available=True)

    # Get the reviews
    reviews = None
    for product in products:
        reviews = ReviewRating.objects.filter(product_id=product.id, status=True)

    context = {
        'products': products,
        'reviews': reviews,
    }
    return render(request, 'home.html', context)
```

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

urlpatterns = [
    path('securelogin/', admin.site.urls),
    path('', views.home, name='home'),
    path('store/', include('store.urls')),
    path('cart/', include('carts.urls')),
    path('accounts/', include('accounts.urls')),

    # ORDERS
    path('orders/', include('orders.urls')),
] + static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)
```

Store Page:

Functionality:

Renders products details in a responsive layout using Bootstrap grid system.

Technologies Used:

Frontend: HTML, CSS, Bootstrap

Backend: Django

Views.py

```
from django.shortcuts import render, get_object_or_404, redirect
from .models import Product, ReviewRating, ProductGallery
from category.models import Category
from carts.models import CartItem
from django.db.models import Q

from carts.views import _cart_id
```

```python
from django.core.paginator import EmptyPage, PageNotAnInteger, Paginator
from django.http import HttpResponse
from .forms import ReviewForm
from django.contrib import messages
from orders.models import OrderProduct


def store(request, category_slug=None):
    categories = None
    products = None

    if category_slug != None:
        categories = get_object_or_404(Category, slug=category_slug)
        products = Product.objects.filter(category=categories, is_available=True)
        paginator = Paginator(products, 1)
        page = request.GET.get('page')
        paged_products = paginator.get_page(page)
        product_count = products.count()
    else:
        products = Product.objects.all().filter(is_available=True).order_by('id')
        paginator = Paginator(products, 3)
        page = request.GET.get('page')
        paged_products = paginator.get_page(page)
        product_count = products.count()

    context = {
        'products': paged_products,
        'product_count': product_count,
    }
    return render(request, 'store/store.html', context)


def product_detail(request, category_slug, product_slug):
    try:
        single_product = Product.objects.get(category__slug=category_slug,
slug=product_slug)
        in_cart = CartItem.objects.filter(cart__cart_id=_cart_id(request),
product=single_product).exists()
    except Exception as e:
        raise e

    if request.user.is_authenticated:
        try:
            orderproduct = OrderProduct.objects.filter(user=request.user,
product_id=single_product.id).exists()
```

```
        except OrderProduct.DoesNotExist:
            orderproduct = None
    else:
        orderproduct = None

    # Get the reviews
    reviews = ReviewRating.objects.filter(product_id=single_product.id,
status=True)

    # Get the product gallery
    product_gallery = ProductGallery.objects.filter(product_id=single_product.id)

    context = {
        'single_product': single_product,
        'in_cart'       : in_cart,
        'orderproduct': orderproduct,
        'reviews': reviews,
        'product_gallery': product_gallery,
    }
    return render(request, 'store/product_detail.html', context)


def search(request):
    if 'keyword' in request.GET:
        keyword = request.GET['keyword']
        if keyword:
            products = Product.objects.order_by('-
created_date').filter(Q(description__icontains=keyword) |
Q(product_name__icontains=keyword))
            product_count = products.count()
    context = {
        'products': products,
        'product_count': product_count,
    }
    return render(request, 'store/store.html', context)


def submit_review(request, product_id):
    url = request.META.get('HTTP_REFERER')
    if request.method == 'POST':
        try:
            reviews = ReviewRating.objects.get(user__id=request.user.id,
product__id=product_id)
            form = ReviewForm(request.POST, instance=reviews)
            form.save()
```

```
            messages.success(request, 'Thank you! Your review has been updated.')
            return redirect(url)
        except ReviewRating.DoesNotExist:
            form = ReviewForm(request.POST)
            if form.is_valid():
                data = ReviewRating()
                data.subject = form.cleaned_data['subject']
                data.rating = form.cleaned_data['rating']
                data.review = form.cleaned_data['review']
                data.ip = request.META.get('REMOTE_ADDR')
                data.product_id = product_id
                data.user_id = request.user.id
                data.save()
                messages.success(request, 'Thank you! Your review has been
submitted.')
                return redirect(url)
```

Models

```
from django.db import models
from category.models import Category
from django.urls import reverse
from accounts.models import Account
from django.db.models import Avg, Count

# Create your models here.

class Product(models.Model):
    product_name    = models.CharField(max_length=200, unique=True)
    slug            = models.SlugField(max_length=200, unique=True)
    description     = models.TextField(max_length=500, blank=True)
    price           = models.IntegerField()
    images          = models.ImageField(upload_to='photos/products')
    stock           = models.IntegerField()
    is_available    = models.BooleanField(default=True)
    category        = models.ForeignKey(Category, on_delete=models.CASCADE)
    created_date    = models.DateTimeField(auto_now_add=True)
    modified_date   = models.DateTimeField(auto_now=True)

    def get_url(self):
        return reverse('product_detail', args=[self.category.slug, self.slug])

    def __str__(self):
        return self.product_name

    def averageReview(self):
        reviews = ReviewRating.objects.filter(product=self,
status=True).aggregate(average=Avg('rating'))
        avg = 0
        if reviews['average'] is not None:
            avg = float(reviews['average'])
        return avg

    def countReview(self):
        reviews = ReviewRating.objects.filter(product=self,
status=True).aggregate(count=Count('id'))
        count = 0
        if reviews['count'] is not None:
            count = int(reviews['count'])
        return count

class VariationManager(models.Manager):
```

```
    def colors(self):
        return super(VariationManager, self).filter(variation_category='color',
is_active=True)

    def sizes(self):
        return super(VariationManager, self).filter(variation_category='size',
is_active=True)

variation_category_choice = (
    ('color', 'color'),
    ('size', 'size'),
)

class Variation(models.Model):
    product = models.ForeignKey(Product, on_delete=models.CASCADE)
    variation_category = models.CharField(max_length=100,
choices=variation_category_choice)
    variation_value     = models.CharField(max_length=100)
    is_active           = models.BooleanField(default=True)
    created_date        = models.DateTimeField(auto_now=True)

    objects = VariationManager()

    def __str__(self):
        return self.variation_value


class ReviewRating(models.Model):
    product = models.ForeignKey(Product, on_delete=models.CASCADE)
    user = models.ForeignKey(Account, on_delete=models.CASCADE)
    subject = models.CharField(max_length=100, blank=True)
    review = models.TextField(max_length=500, blank=True)
    rating = models.FloatField()
    ip = models.CharField(max_length=20, blank=True)
    status = models.BooleanField(default=True)
    created_at = models.DateTimeField(auto_now_add=True)
    updated_at = models.DateTimeField(auto_now=True)

    def __str__(self):
        return self.subject


class ProductGallery(models.Model):
    product = models.ForeignKey(Product, default=None, on_delete=models.CASCADE)
    image = models.ImageField(upload_to='store/products', max_length=255)

    def __str__(self):
```

```
        return self.product.product_name

    class Meta:
        verbose_name = 'productgallery'
        verbose_name_plural = 'product gallery'
```

Cart Page:

Functionality:

Shows products added by user that he/she wants to purchase are sent to cart.

From cart user and add and remove product also.

Technologies Used:

Frontend: HTML, CSS, JavaScript

Backend: Django, Django Admin

Model Fields:

User
Product
Cart
Quantity
Is active
variation

urls.py

```
from django.urls import path
from . import views


urlpatterns = [
    path('', views.cart, name='cart'),
    path('add_cart/<int:product_id>/', views.add_cart, name='add_cart'),
```

```
    path('remove_cart/<int:product_id>/<int:cart_item_id>/', views.remove_cart,
name='remove_cart'),
    path('remove_cart_item/<int:product_id>/<int:cart_item_id>/',
views.remove_cart_item, name='remove_cart_item'),

    path('checkout/', views.checkout, name='checkout'),
]
```

Orders Page:

Functionality:

This page is displayed while placing order and making payment.

Forms integrated with Django models for seamless data management.

Technologies Used:

Frontend: HTML, CSS, JavaScript

Backend: Django, Django Forms, Django Models

Forms.py

```
from django import forms
from .models import Order


class OrderForm(forms.ModelForm):
    class Meta:
        model = Order
        fields = ['first_name', 'last_name', 'phone', 'email',
'address_line_1', 'address_line_2', 'country', 'state', 'city',
'order_note']
```

Accounts:

Functionality:

This section manages all the login, email verification, dashboard related information.

Technologies Used:

```
Frontend: HTML, CSS, Bootstrap
```

urls.py

```
from django.urls import path
from . import views


urlpatterns = [
    path('register/', views.register, name='register'),
    path('login/', views.login, name='login'),
    path('logout/', views.logout, name='logout'),
    path('dashboard/', views.dashboard, name='dashboard'),
    path('', views.dashboard, name='dashboard'),

    path('activate/<uidb64>/<token>/', views.activate, name='activate'),
    path('forgotPassword/', views.forgotPassword, name='forgotPassword'),
```

```
    path('resetpassword_validate/<uidb64>/<token>/',
views.resetpassword_validate, name='resetpassword_validate'),
    path('resetPassword/', views.resetPassword, name='resetPassword'),

    path('my_orders/', views.my_orders, name='my_orders'),
    path('edit_profile/', views.edit_profile, name='edit_profile'),
    path('change_password/', views.change_password,
name='change_password'),
    path('order_detail/<int:order_id>/', views.order_detail,
name='order_detail'),
]
```

6. Challenges Faced:

1.Integration of smtp using gmail for email verification and paypal for making payments and invoice generation. This involved configuring Django's admin and authentication with aforesaid technology.

2.Customisation of dashboard made. It was a tedious task as the id and forms and templates all rendered took a lot more of effort than expected.

7. Future Enhancements:

1.Integration of PayPal account for payment purposes.

2.Implementation of user authentication and personalized user experiences.

3.Enhanced Cart features and also made product review form.

4.Integration with smtp gmail services for email verification, activation and order confirmation.

8. Conclusion:

The TrendyKraft project successfully delivers a user-friendly shopping experience to users using Django framework. Its modular design, integration of various technologies, and focus on user experience make it a valuable tool for shopping enthusiast.

9. References:

1.Django Documentation: [https://docs.djangoproject.com/en/5.0/]

2.Bootstrap Documentation: [https://getbootstrap.com/docs/4.1/getting-started/introduction/

3. Python- Decouple [https://pypi.org/project/python-decouple/]

4. JQuery - https://jquery.com/

5. PayPal - https://developer.paypal.com/docs/checkout/standard/integrate/