CHAPTER 1

INTRODUCTION

The dynamic consumer electronics market of today presents several problems for an efficient repair shop manager. Due to the lack of a centralized system to track product availability, warranty durations, and repair schedules, the existing procedure is inefficient and unsatisfactory to customers. The inability of repair professionals to view work statuses in real-time and their frequent struggles with manual record-keeping impede timely service delivery and have an adverse effect on business growth. To address these challenges, this e-commerce project leverages modern web technologies to create a robust and user-friendly platform, streamlining operations for both customers and repair professionals.

This e-commerce platform, developed using Django, PostgreSQL, HTML, CSS, and JavaScript, is designed to offer a seamless and efficient shopping experience for consumer electronics. The project encompasses various features such as user registration and authentication, comprehensive product browsing, detailed product information, and a streamlined checkout process. By integrating a secure payment gateway and providing real-time order confirmation and tracking, the platform ensures a reliable and satisfying experience for customers. Additionally, the website's responsive design guarantees accessibility across multiple devices, catering to the diverse needs of modern consumers.

The backend, built with Django and PostgreSQL, ensures robust data management and scalability. Django's ORM simplifies database interactions, allowing efficient data retrieval and manipulation while maintaining relational integrity. The frontend, crafted with HTML, CSS, and JavaScript, offers a visually appealing and interactive user interface. Key features include dynamic product listings, an intuitive shopping cart, and secure checkout. Performance optimizations, such as caching strategies and database indexing, enhance the platform's speed and reliability. Security measures, including CSRF protection and input validation, safeguard user data and protect against common web vulnerabilities.

1.1 Existing System

In the current landscape of consumer electronics repair, many businesses face inefficiencies due to the absence of a centralized system. Repair shops often struggle with tracking product availability, managing warranty periods, and scheduling repairs, which are typically done manually. This manual approach results in delays, errors, and a lack of transparency, leading to customer dissatisfaction and stunted business growth.

1.2. Problem Identification

The lack of real-time data access for repair professionals, coupled with cumbersome manual record-keeping, hampers their ability to deliver timely and effective service. The absence of an integrated system

1.3 Feasibility and Usefulness to Society

1.Centralized System Implementation

- **Feasibility:** Leveraging existing web technologies like Django and PostgreSQL, it is feasible to develop a robust and scalable centralized system that can handle the diverse needs of repair shops.
- **Usefulness:** A centralized system will streamline operations, reducing manual errors and improving service efficiency, which is critical in today's competitive market.

2. Real-Time Data Accessibility

- **Feasibility:** Integrating real-time data tracking is feasible with modern web frameworks and database management systems.
- **Usefulness:** Real-time data accessibility enhances decision-making, allows for better resource allocation, and improves customer satisfaction by providing timely updates.

3. Automation of Routine Tasks

- **Feasibility:** Automation of inventory management, job scheduling, and warranty tracking can be easily implemented using Django's ORM and background task management tools.
- **Usefulness:** Automating routine tasks reduces operational overhead, minimizes human error, and frees up time for repair professionals to focus on more complex tasks.

4. Secure Transactions and Data Management

- **Feasibility:** Implementing secure payment gateways and ensuring data protection with Django's built-in security features is both feasible and effective.
- **Usefulness:** Ensuring secure transactions and data management builds customer trust, which is vital for retaining clients and growing the business.

5. Scalability and Long-Term Viability

- **Feasibility:** The platform’s architecture is designed to scale as the business grows, accommodating increasing user and transaction volumes without significant reengineering.
- **Usefulness:** Scalability ensures the long-term viability of the platform, allowing it to grow alongside the business and continue to meet the evolving needs of both customers and repair professionals. to manage inventory, track warranties, and schedule repairs exacerbates these challenges, making the entire repair process inefficient and unreliable.

CHAPTER 2

PROBLEM STATEMENT

Smart Gadget Repair Solutions The dynamic consumer electronics market of today presents several problems for an efficient repair shop manager. Due to the lack of a centralized system to track product availability, warranty durations, and repair schedules, the existing procedure is inefficient and unsatisfactory to customers. The inability of repair professionals to view work statuses in real time and their frequent struggles with manual record-keeping impede timely service delivery and have an adverse effect on business growth.

1. **Product Availability**
    1. Real-time Inventory Tracking: Monitor the availability of parts and products in real-time.
    2. Automatic Restocking Alerts: Get notifications when inventory levels are low.
    3. Supplier Integration: Connect with suppliers for seamless ordering and tracking of parts and products.
    4. Product Catalog: Maintain a detailed catalog of all available products and parts with descriptions and images.
2. **Time Period to Repair**
    1. Job Scheduling and Tracking: Assign and track repair jobs with estimated completion times.
    2. Technician Availability: Check the availability of technicians and assign jobs based on their expertise and workload.
    3. Progress Updates: Provide customers with real-time updates on the status of their repairs.
    4. Repair Time Estimation: Automatically estimate the time required for repairs based on the type of issue and technician's experience.
    5. Service Queue Management: Manage and prioritize repair jobs based on urgency and customer preferences.
3. **Warranty Period**
    1. Warranty Tracking: Maintain records of warranty periods for all repaired items.
    2. Warranty Validation: Check the warranty status of products when they are brought in for repair.
    3. Automated Warranty Notifications: Send reminders to customers about the expiration of their warranty periods.

   4. Warranty Claims Management: Simplify the process for customers to claim warranties, including documentation and approval workflows.
   5. Historical Warranty Data: Access past warranty information and repair history for better service and support.

4. **Customer Relationship Management (CRM)**
   1. Customer Profiles: Maintain detailed profiles with contact information, repair history, and preferences.
   2. Communication History: Track all communications with customers, including emails, calls, and messages.
   3. Feedback and Reviews: Collect and analyze customer feedback and reviews to improve services
5. **Billing and Payments**
   1. Invoicing: Generate detailed invoices for repairs and services.
   2. Payment Processing: Support multiple payment methods, including credit/debit cards, mobile payments, and online transfers.
   3. Billing History: Maintain a history of all transactions for easy reference and accounting.
6. **Reporting and Analytics**
   1. Repair Metrics: Track key metrics such as repair times, technician performance, and customer satisfaction.
   2. Financial Reports: Generate reports on revenue, expenses, and profitability.
   3. Inventory Reports: Analyse inventory usage and trends to optimize stock levels.
7. **User Management and Security**
   1. Role-Based Access Control: Manage user permissions based on roles and responsibilities.
   2. Data Security: Ensure the security of customer data and repair records with encryption and secure access protocols.

2.1 Project Relevance Based on title

Centralized System for Repair Management

Relevance: The project directly addresses the need for a centralized system to manage repair shop operations, tackling inefficiencies caused by manual processes and fragmented record-keeping. By consolidating data on product availability, warranty periods, and repair schedules, the system enhances operational efficiency.

Real-Time Inventory and Job Tracking

Relevance: The lack of real-time access to inventory and repair job statuses is a critical issue highlighted in the problem statement. The project's focus on implementing real-time tracking ensures that repair professionals can monitor and manage their workload more effectively, leading to faster service delivery and improved customer satisfaction.

Automated Warranty and Repair Scheduling

Relevance: The challenge of managing warranty periods and repair schedules manually often results in delays and customer dissatisfaction. The project addresses this by introducing automation for warranty tracking and job scheduling, ensuring timely repairs and better service management.

Customer Relationship Management (CRM) Integration

Relevance: Maintaining strong customer relationships is crucial for business growth, as identified in the problem statement. The project's inclusion of CRM features like customer profiles, communication tracking, and feedback collection directly responds to the need for enhanced customer engagement and satisfaction.

Secure Billing and Payment Processing

Relevance: The problem statement emphasizes the importance of efficient billing and payment processes. The project's focus on secure invoicing and multi-method payment processing ensures that transactions are smooth and secure, which is essential for customer trust and business credibility.

Comprehensive Reporting and Analytics

Relevance: Tracking repair metrics, financial performance, and inventory trends is crucial for informed decision-making. The project's reporting and analytics capabilities provide repair shop managers with the insights needed to optimize operations, reduce costs, and enhance profitability, aligning closely with the issues identified in the problem statement.

CHAPTER 3

OBJECTIVES

The objectives for this project can include:

1. User-Friendly Interface: Develop a website with an intuitive and easy-to-navigate interface to enhance the user experience.

2. Efficient Registration/Login: Implement a seamless registration and login process to reduce barriers for new users and facilitate quick access for returning users.

3. Comprehensive Product Browsing: Provide robust product browsing capabilities, including search, filtering, and sorting options to help users find products easily.

4. Detailed Product Information: Ensure that detailed and accurate product information is available to help users make informed purchasing decisions.

5. Smooth Shopping Cart Experience: Create a streamlined process for adding items to the cart, viewing the cart, and making adjustments as needed.

6. Secure Checkout Process: Develop a secure and efficient checkout process that minimizes friction and ensures the safety of user payment information.

7. Reliable Payment Gateway: Integrate a reliable and secure payment gateway to handle transactions efficiently and securely.

8. Order Confirmation and Tracking: Provide immediate order confirmation and the ability for users to track their orders post-purchase.

9. Wishlist Management: Offer users the ability to manage their Wishlist, enabling them to save products for future consideration.

10. Account Security and Privacy: Implement robust security measures to protect user data and ensure privacy.

11. Responsive Design: Ensure the website is fully responsive, providing an optimal experience across all devices, including desktops, tablets, and smartphones.

12. Performance Optimization: Optimize website performance to ensure fast loading times and smooth interactions.

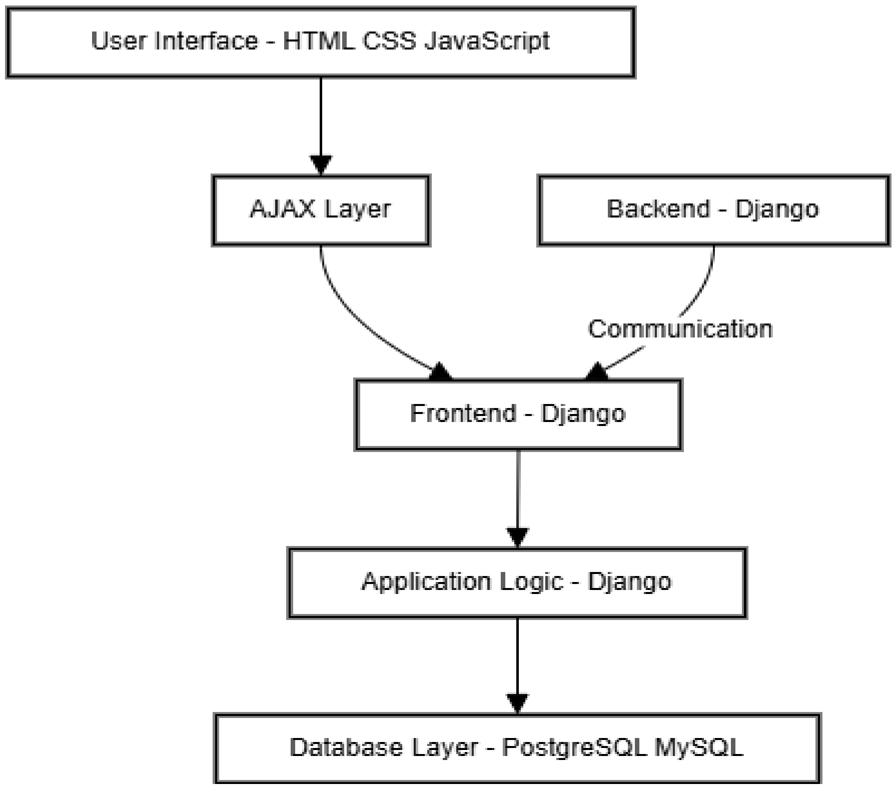
13. Scalability: Design the website architecture to be scalable to handle increasing traffic and data as the business grows.

14. Customer Support Integration: Integrate customer support features, such as chatbots or live chat, to assist users with their inquiries and issues.

15. Analytics and Reporting: Implement analytics tools to track user behavior, sales, and other key metrics to inform business decisions and improve the website.

CHAPTER 4

ARCHITECTURAL DESIGN


User Interface - HTML CSS JavaScript
AJAX Layer
Backend - Django
Communication
Frontend - Django
Application Logic - Django
Database Layer - PostgreSQL MySQL

This system architecture is designed to create a robust and efficient e-commerce platform for managing consumer electronics repairs. The platform is built using a combination of modern web technologies, with Django being used for both the frontend and backend, ensuring a seamless integration of all components.

1. User Interface (HTML, CSS, JavaScript)

- **Role**: The User Interface (UI) layer is the first point of interaction for users. It is designed using HTML for structure, CSS for styling, and JavaScript for dynamic behavior. This layer is responsible for presenting the data and functionalities in an intuitive and user-friendly manner.
- **Function**: Users can interact with various features like browsing products, checking repair statuses, and making payments through this interface.

2. AJAX Layer

- **Role**: AJAX (Asynchronous JavaScript and XML) is used to enhance the user experience by enabling asynchronous communication between the user interface and the server.

- **Function**: This allows for real-time updates without needing to reload the entire page, which is particularly useful for features like live inventory updates and dynamic form submissions.

3. Frontend (Django)

- **Role**: The frontend is managed by Django, which is a high-level Python web framework. In this architecture, Django serves both the frontend and backend, ensuring tight integration and consistency across the platform.
- **Function**: Django renders the HTML templates and serves the web pages to the users. It also handles the routing of requests and responses, ensuring that the right data is delivered to the UI.

4. Backend (Django)

- **Role**: The backend is also powered by Django, handling all the core application logic, data processing, and business rules.
- **Function**: This layer is responsible for processing user requests, interacting with the database, managing sessions, and applying security measures. Django's ORM (Object-Relational Mapping) simplifies database interactions, making it easier to perform CRUD (Create, Read, Update, Delete) operations.

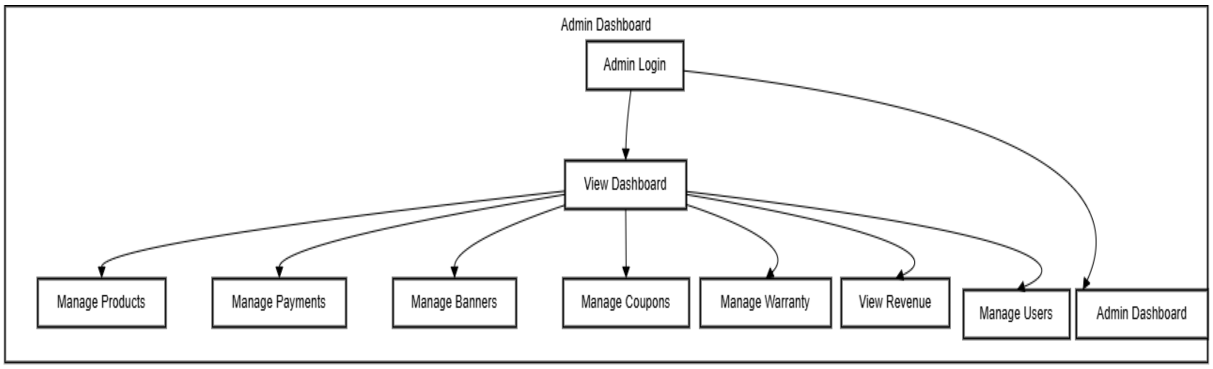
5. Application Logic (Django)

- **Role**: The application logic is where all the business rules and processes are implemented. It manages how data flows between the frontend and the database, and how different components of the platform interact with each other.
- **Function**: This layer ensures that all functionalities, such as user authentication, product management, repair scheduling, and order processing, work as intended and meet the business requirements.

6. Database Layer (PostgreSQL/MySQL)

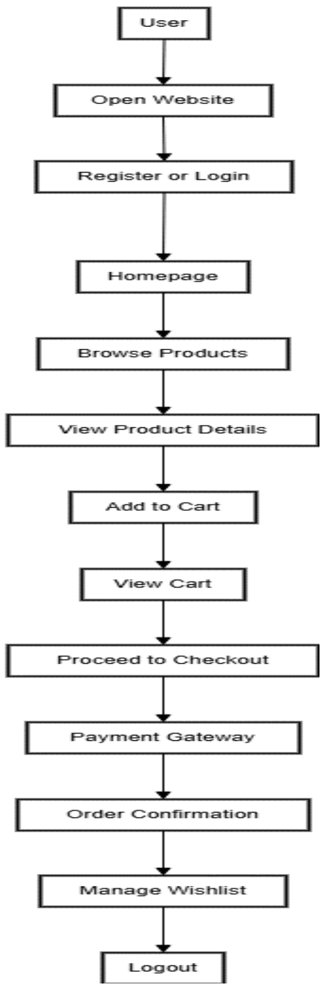
- **Role**: The database layer is where all the platform's data is stored and managed. PostgreSQL or MySQL can be used depending on the specific needs for scalability, performance, and data integrity.
- **Function**: The database stores critical information such as user details, product inventories, repair histories, and transaction records. The Django ORM interacts with the database to perform queries and updates in a structured and efficient manner.

4.1Data flow of ADMIN Module



1. **Admin Login:** The entry point where the admin logs into the system.
2. **View Dashboard:** After logging in, the admin is directed to the main dashboard.
3. **Admin Dashboard:** This serves as both an entry point and a destination for other functionalities. From the main dashboard, the admin can perform various management tasks.
4. **Management Tasks:**
   - **Manage Products:** The admin can manage product-related information.
   - **Manage Payments:** The admin can oversee and handle payment transactions.
   - **Manage Banners:** The admin can control the banners displayed within the system.
   - **Manage Coupons:** The admin can create, update, or delete coupons.
   - **Manage Warranty:** The admin can manage warranty details for products.
   - **View Revenue:** The admin can view revenue statistics and reports.
   - **Manage Users:** The admin can manage user accounts and their details.
   - **Back to Admin Dashboard:** Each of these tasks allows navigation back to the main dashboard.

4.2Data Flow of User Module



1. User: The starting point where the user initiates interaction.

2. Open Website: The user opens the e-commerce website.

3. Register or Login: The user either registers for a new account or logs into an existing account.

4. Homepage: After logging in or registering, the user is directed to the homepage.

5. Browse Products: The user navigates through the website to browse available products.

6. View Product Details: When the user finds a product of interest, they view the detailed information about that product.

7. Add to Cart: The user adds the product to their shopping cart.

8. View Cart: The user views the items in their shopping cart.

9. Proceed to Checkout: The user proceeds to the checkout process to purchase the items in their cart.

10. Payment Gateway: The user enters payment information and completes the transaction through the payment gateway.

11. Order Confirmation: The user receives an order confirmation, indicating that the purchase was successful.

12. Manage Wishlist: The user can manage their Wishlist, adding or removing items they are interested in for future purchases.

13. Logout: Finally, the user logs out of their account.

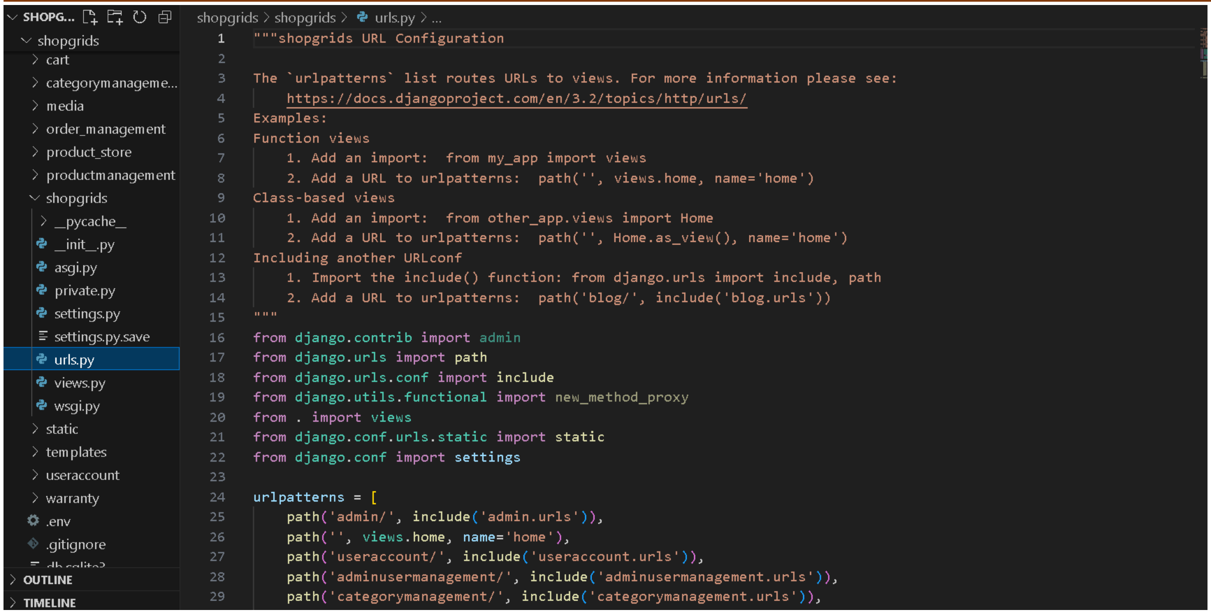
CHAPTER 5

IMPLEMENTATION

The implementation of this e-commerce project involved setting up a Django web framework with PostgreSQL as the backend database, and HTML, CSS, and JavaScript for the frontend interface. The project began with the creation of a Django project and multiple apps to modularize functionalities such as user authentication, product management, and order processing. The database schema was designed using Django's ORM to define models representing users, products, orders, and shopping carts, ensuring relational integrity and efficient data retrieval. The views handled HTTP requests and responses, connecting the frontend templates with backend logic, while URL routing facilitated navigation between different pages. HTML templates were developed for key user interfaces, including the homepage, product detail pages, shopping cart, and checkout process, with CSS for styling and JavaScript for interactive elements like real-time cart updates.

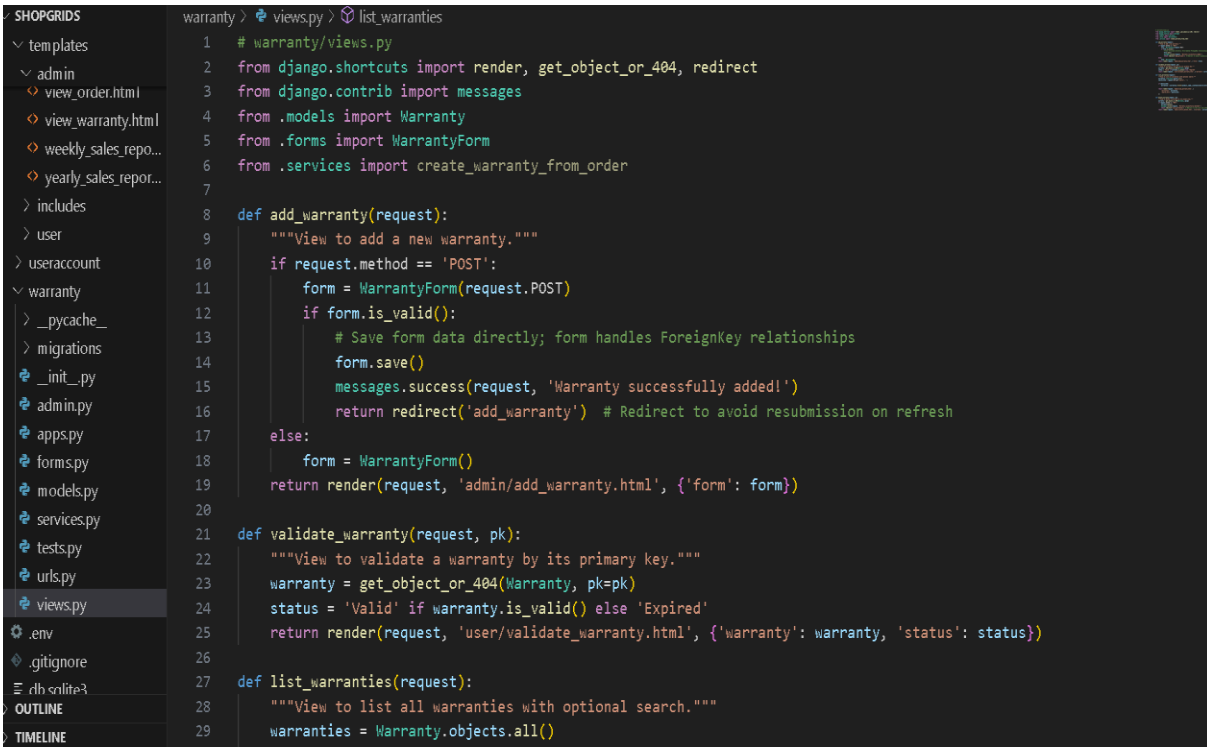
On the frontend, a responsive design was implemented to ensure compatibility across various devices, enhancing user experience. User authentication was secured with Django's built-in authentication system, incorporating features like password hashing and session management. The checkout process integrated with a secure payment gateway to handle transactions, while order confirmation and tracking functionalities provided users with real-time updates on their purchases. Performance optimizations were achieved through caching strategies and database indexing, while security measures included CSRF protection and input validation to guard against common web vulnerabilities. The project was thoroughly tested with unit tests for individual components and integration tests to ensure seamless interaction between different parts of the application. Deployment to a production environment involved configuring a web server and setting up continuous integration and deployment pipelines for automated updates.

5.1 Code Snippet:



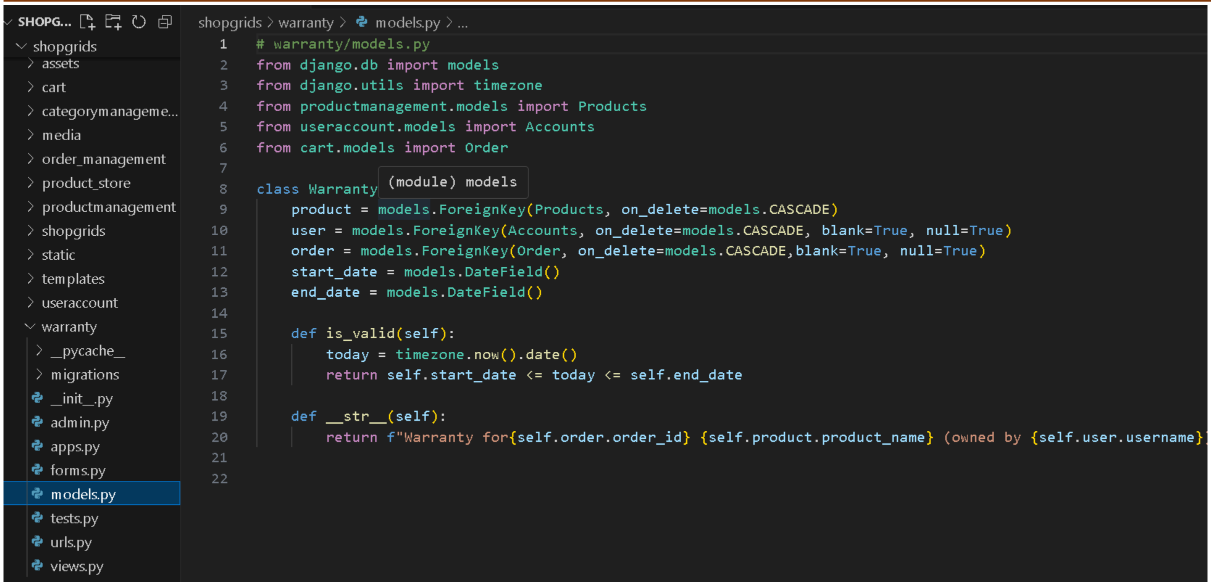
```
"""shopgrids URL Configuration

The `urlpatterns` list routes URLs to views. For more information please see:
    https://docs.djangoproject.com/en/3.2/topics/http/urls/
Examples:
Function views
    1. Add an import:  from my_app import views
    2. Add a URL to urlpatterns:  path('', views.home, name='home')
Class-based views
    1. Add an import:  from other_app.views import Home
    2. Add a URL to urlpatterns:  path('', Home.as_view(), name='home')
Including another URLconf
    1. Import the include() function: from django.urls import include, path
    2. Add a URL to urlpatterns:  path('blog/', include('blog.urls'))
"""
from django.contrib import admin
from django.urls import path
from django.urls.conf import include
from django.utils.functional import new_method_proxy
from . import views
from django.conf.urls.static import static
from django.conf import settings

urlpatterns = [
    path('admin/', include('admin.urls')),
    path('', views.home, name='home'),
    path('useraccount/', include('useraccount.urls')),
    path('adminusermanagement/', include('adminusermanagement.urls')),
    path('categorymanagement/', include('categorymanagement.urls')),
```

Fig 5.1.1 Project Urls.py



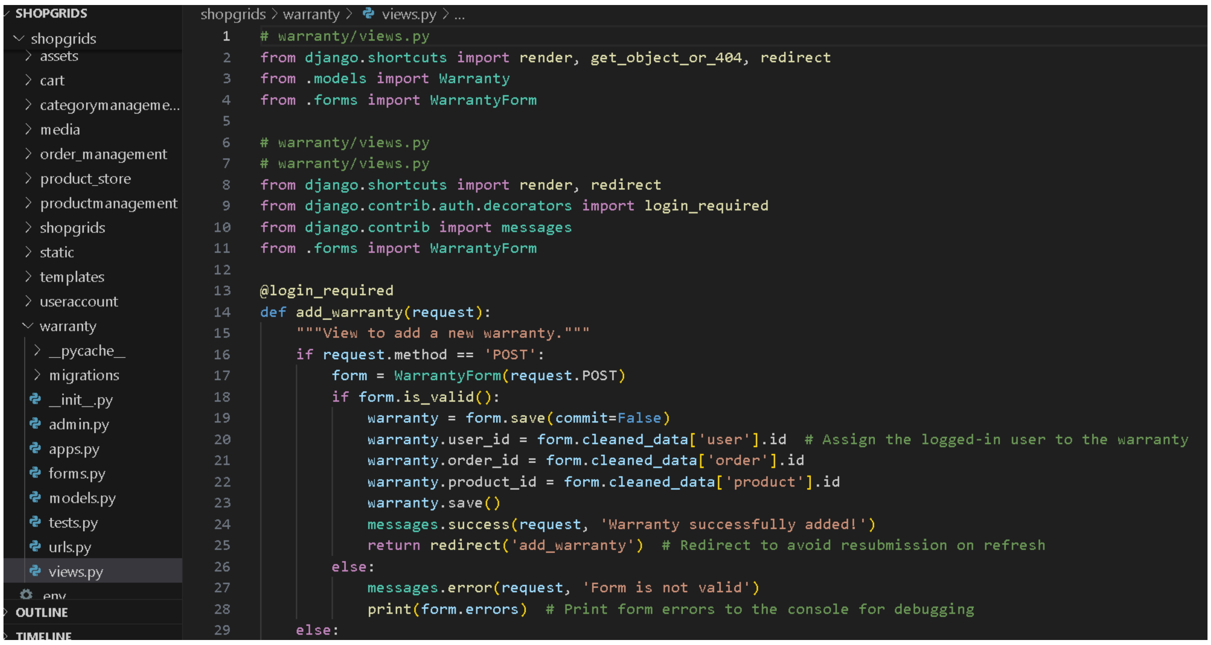
```
# warranty/views.py
from django.shortcuts import render, get_object_or_404, redirect
from django.contrib import messages
from .models import Warranty
from .forms import WarrantyForm
from .services import create_warranty_from_order

def add_warranty(request):
    """View to add a new warranty."""
    if request.method == 'POST':
        form = WarrantyForm(request.POST)
        if form.is_valid():
            # Save form data directly; form handles ForeignKey relationships
            form.save()
            messages.success(request, 'Warranty successfully added!')
            return redirect('add_warranty')  # Redirect to avoid resubmission on refresh
    else:
        form = WarrantyForm()
    return render(request, 'admin/add_warranty.html', {'form': form})

def validate_warranty(request, pk):
    """View to validate a warranty by its primary key."""
    warranty = get_object_or_404(Warranty, pk=pk)
    status = 'Valid' if warranty.is_valid() else 'Expired'
    return render(request, 'user/validate_warranty.html', {'warranty': warranty, 'status': status})

def list_warranties(request):
    """View to list all warranties with optional search."""
    warranties = Warranty.objects.all()
```

Fig 5.1.2 Warranty app/views.py



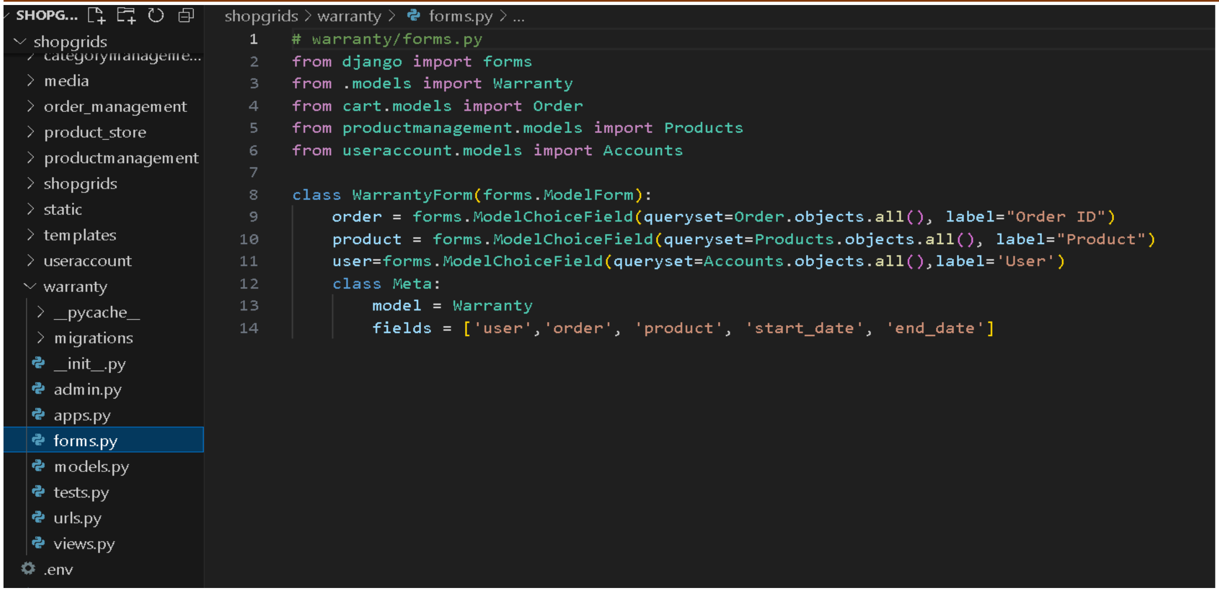
```python
# warranty/models.py
from django.db import models
from django.utils import timezone
from productmanagement.models import Products
from useraccount.models import Accounts
from cart.models import Order

class Warranty
    product = models.ForeignKey(Products, on_delete=models.CASCADE)
    user = models.ForeignKey(Accounts, on_delete=models.CASCADE, blank=True, null=True)
    order = models.ForeignKey(Order, on_delete=models.CASCADE,blank=True, null=True)
    start_date = models.DateField()
    end_date = models.DateField()

    def is_valid(self):
        today = timezone.now().date()
        return self.start_date <= today <= self.end_date

    def __str__(self):
        return f"Warranty for{self.order.order_id} {self.product.product_name} (owned by {self.user.username}
```

Fig 5.1.3 Warranty models.py



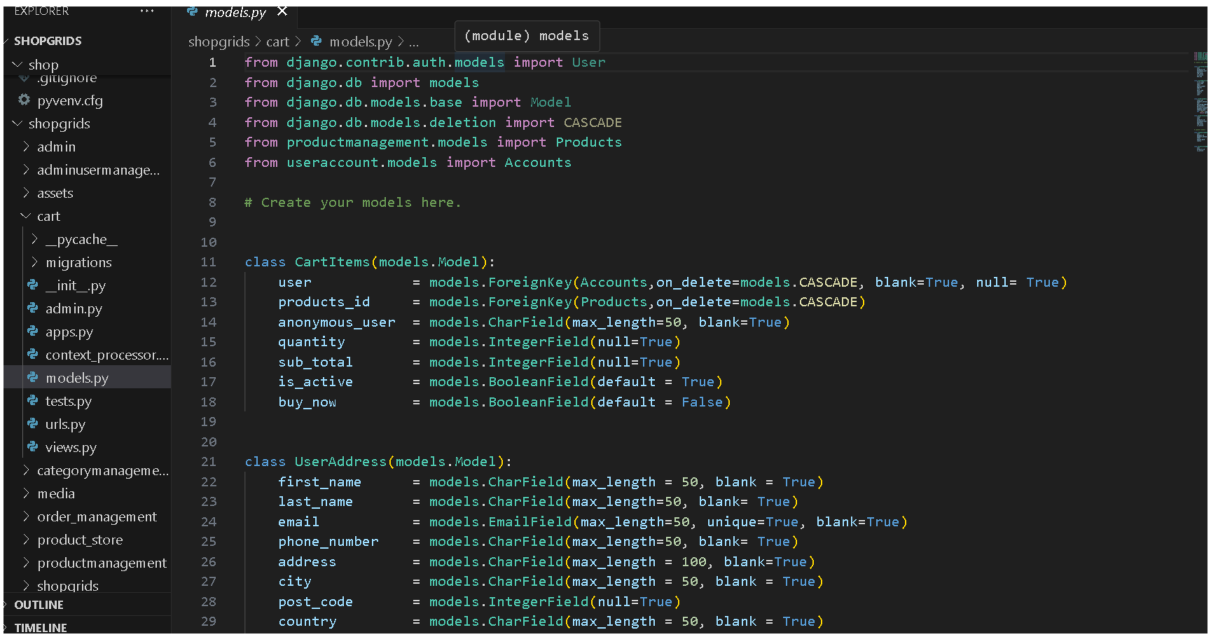
```python
# warranty/views.py
from django.shortcuts import render, get_object_or_404, redirect
from .models import Warranty
from .forms import WarrantyForm

# warranty/views.py
# warranty/views.py
from django.shortcuts import render, redirect
from django.contrib.auth.decorators import login_required
from django.contrib import messages
from .forms import WarrantyForm

@login_required
def add_warranty(request):
    """View to add a new warranty."""
    if request.method == 'POST':
        form = WarrantyForm(request.POST)
        if form.is_valid():
            warranty = form.save(commit=False)
            warranty.user_id = form.cleaned_data['user'].id  # Assign the logged-in user to the warranty
            warranty.order_id = form.cleaned_data['order'].id
            warranty.product_id = form.cleaned_data['product'].id
            warranty.save()
            messages.success(request, 'Warranty successfully added!')
            return redirect('add_warranty')  # Redirect to avoid resubmission on refresh
        else:
            messages.error(request, 'Form is not valid')
            print(form.errors)  # Print form errors to the console for debugging
    else:
```

Fig 5.1.4 Warranty views.py



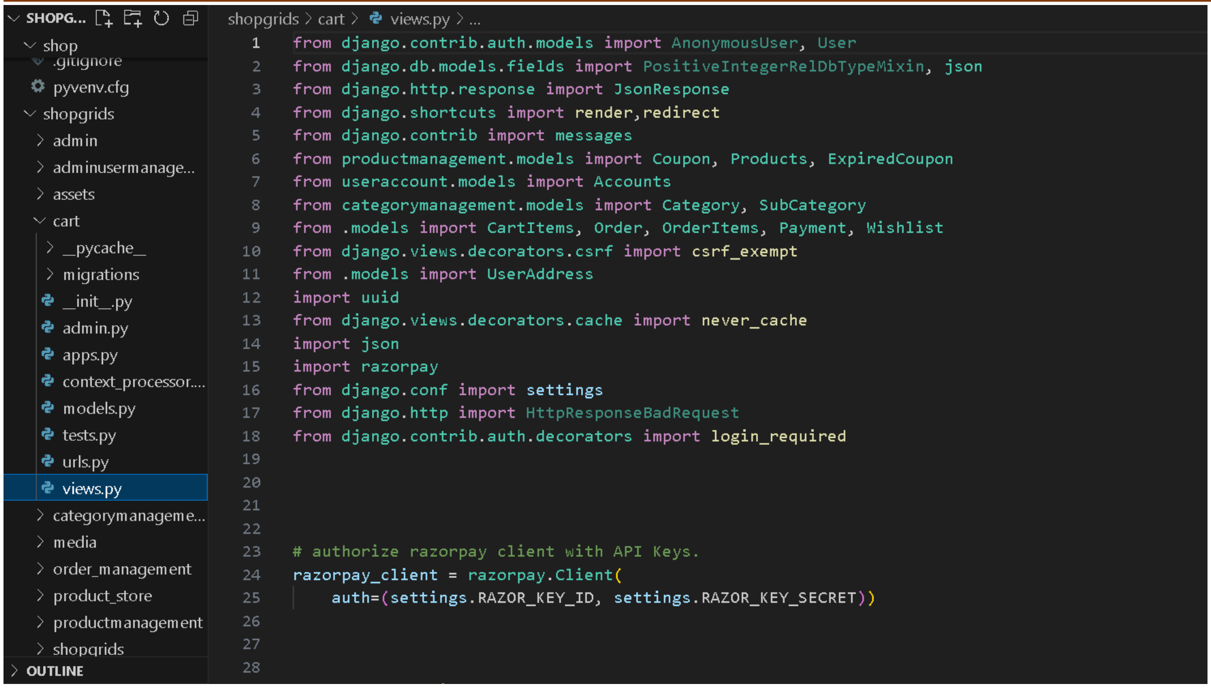
```python
# warranty/forms.py
from django import forms
from .models import Warranty
from cart.models import Order
from productmanagement.models import Products
from useraccount.models import Accounts

class WarrantyForm(forms.ModelForm):
    order = forms.ModelChoiceField(queryset=Order.objects.all(), label="Order ID")
    product = forms.ModelChoiceField(queryset=Products.objects.all(), label="Product")
    user=forms.ModelChoiceField(queryset=Accounts.objects.all(),label='User')
    class Meta:
        model = Warranty
        fields = ['user','order', 'product', 'start_date', 'end_date']
```

Fig 5.1.5 Warranty forms.py



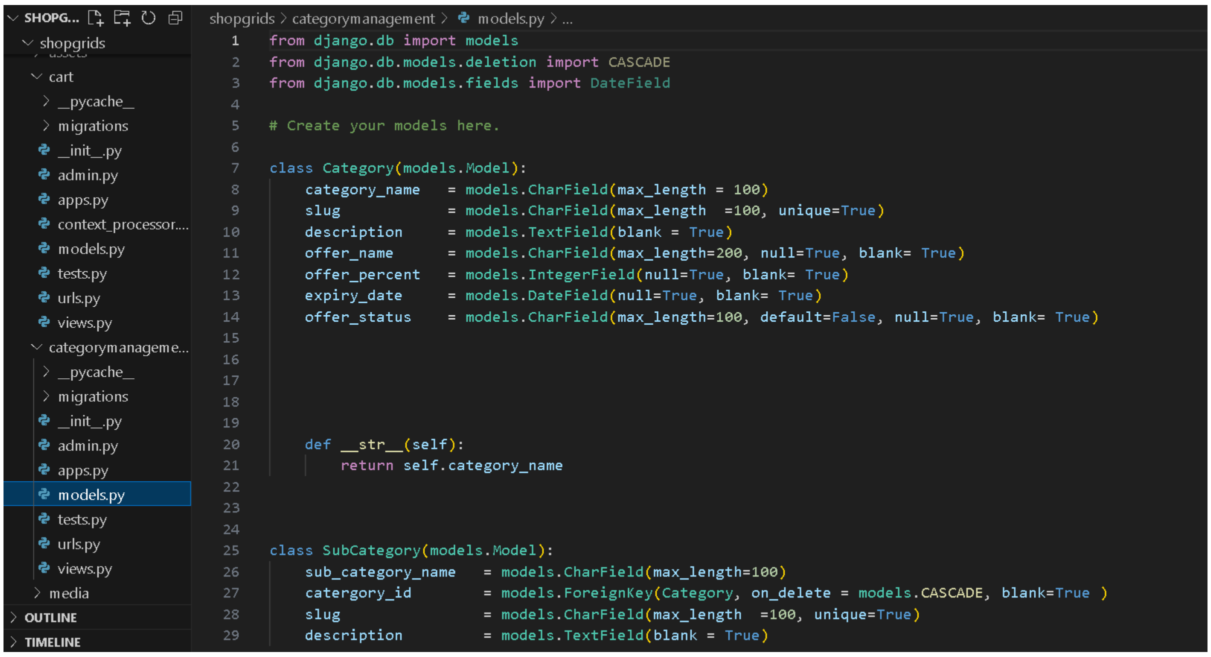
```python
from django.contrib.auth.models import User
from django.db import models
from django.db.models.base import Model
from django.db.models.deletion import CASCADE
from productmanagement.models import Products
from useraccount.models import Accounts

# Create your models here.


class CartItems(models.Model):
    user            = models.ForeignKey(Accounts,on_delete=models.CASCADE, blank=True, null= True)
    products_id     = models.ForeignKey(Products,on_delete=models.CASCADE)
    anonymous_user  = models.CharField(max_length=50, blank=True)
    quantity        = models.IntegerField(null=True)
    sub_total       = models.IntegerField(null=True)
    is_active       = models.BooleanField(default = True)
    buy_now         = models.BooleanField(default = False)


class UserAddress(models.Model):
    first_name      = models.CharField(max_length = 50, blank = True)
    last_name       = models.CharField(max_length=50, blank= True)
    email           = models.EmailField(max_length=50, unique=True, blank=True)
    phone_number    = models.CharField(max_length=50, blank= True)
    address         = models.CharField(max_length = 100, blank=True)
    city            = models.CharField(max_length = 50, blank = True)
    post_code       = models.IntegerField(null=True)
    country         = models.CharField(max_length = 50, blank = True)
```

Fig 5.1.6 Cart models.py



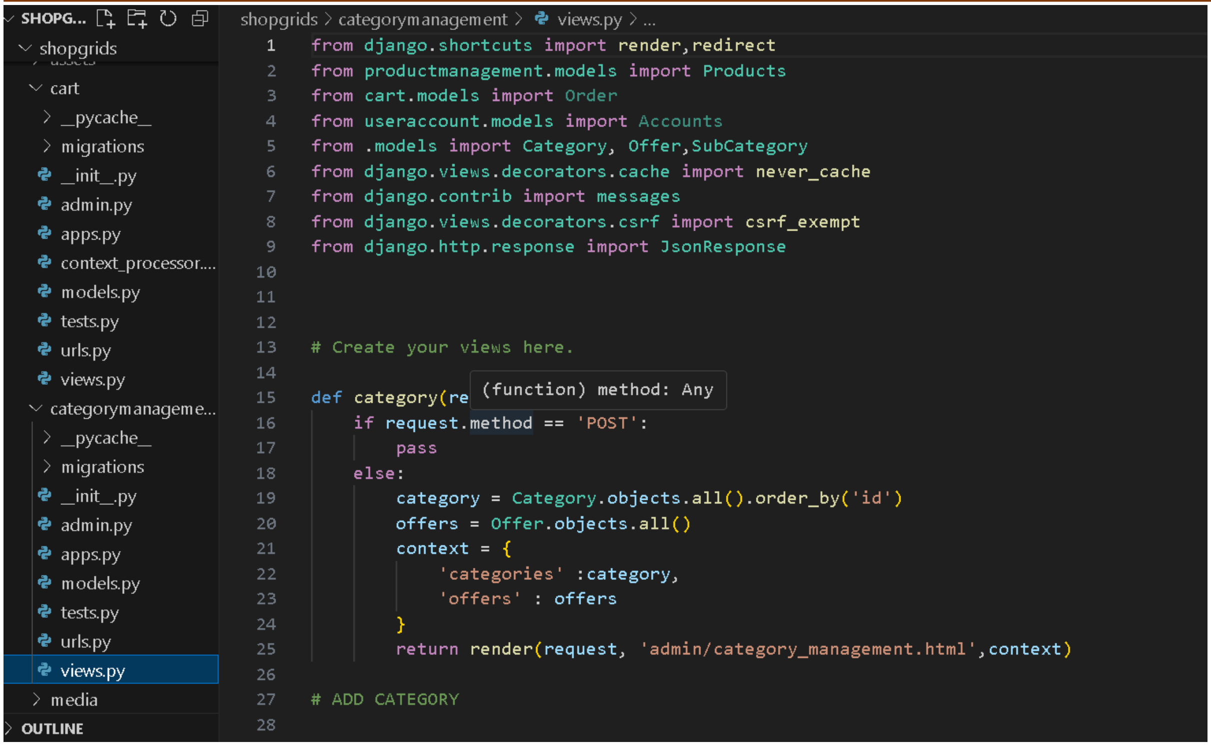
```
from django.contrib.auth.models import AnonymousUser, User
from django.db.models.fields import PositiveIntegerRelDbTypeMixin, json
from django.http.response import JsonResponse
from django.shortcuts import render,redirect
from django.contrib import messages
from productmanagement.models import Coupon, Products, ExpiredCoupon
from useraccount.models import Accounts
from categorymanagement.models import Category, SubCategory
from .models import CartItems, Order, OrderItems, Payment, Wishlist
from django.views.decorators.csrf import csrf_exempt
from .models import UserAddress
import uuid
from django.views.decorators.cache import never_cache
import json
import razorpay
from django.conf import settings
from django.http import HttpResponseBadRequest
from django.contrib.auth.decorators import login_required



# authorize razorpay client with API Keys.
razorpay_client = razorpay.Client(
    auth=(settings.RAZOR_KEY_ID, settings.RAZOR_KEY_SECRET))
```

Fig 5.1.7 Cart views.py



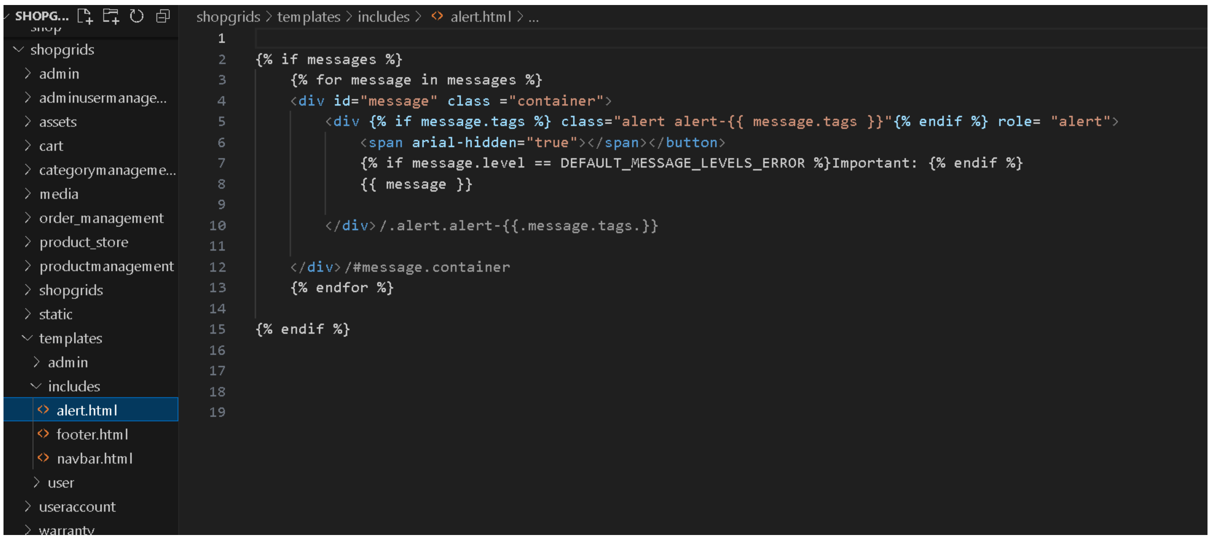
```
from django.db import models
from django.db.models.deletion import CASCADE
from django.db.models.fields import DateField

# Create your models here.

class Category(models.Model):
    category_name   = models.CharField(max_length = 100)
    slug            = models.CharField(max_length  =100, unique=True)
    description     = models.TextField(blank = True)
    offer_name      = models.CharField(max_length=200, null=True, blank= True)
    offer_percent   = models.IntegerField(null=True, blank= True)
    expiry_date     = models.DateField(null=True, blank= True)
    offer_status    = models.CharField(max_length=100, default=False, null=True, blank= True)





    def __str__(self):
        return self.category_name



class SubCategory(models.Model):
    sub_category_name    = models.CharField(max_length=100)
    catergory_id         = models.ForeignKey(Category, on_delete = models.CASCADE, blank=True )
    slug                 = models.CharField(max_length  =100, unique=True)
    description          = models.TextField(blank = True)
```

Fig 5.1.8 Category management models.py



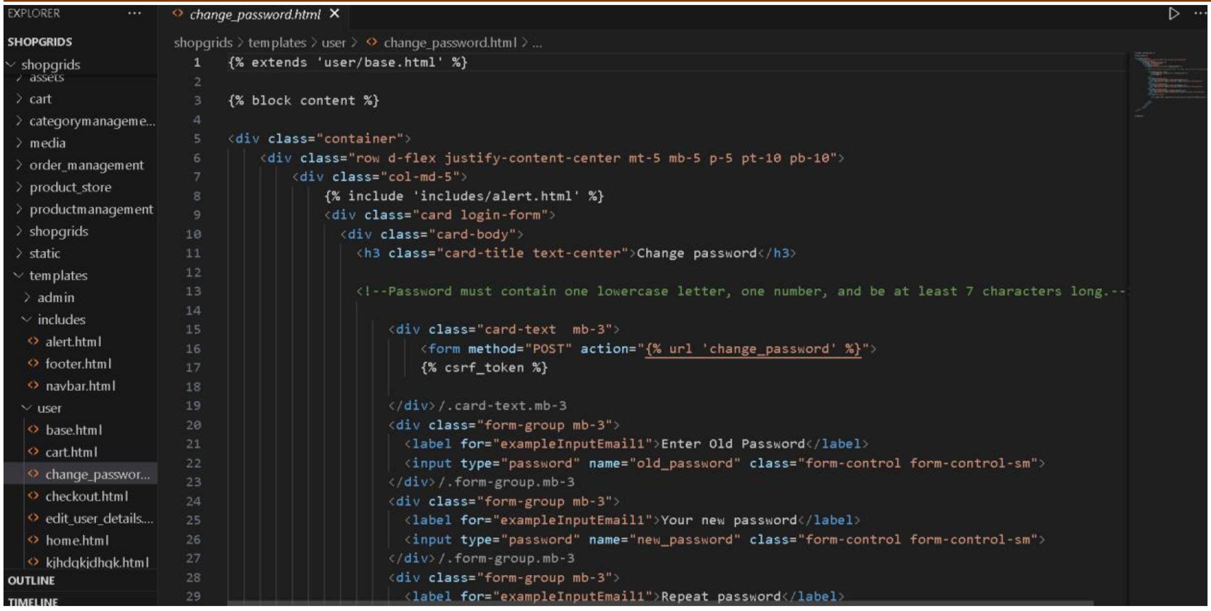
```python
from django.shortcuts import render,redirect
from productmanagement.models import Products
from cart.models import Order
from useraccount.models import Accounts
from .models import Category, Offer,SubCategory
from django.views.decorators.cache import never_cache
from django.contrib import messages
from django.views.decorators.csrf import csrf_exempt
from django.http.response import JsonResponse



# Create your views here.

def category(re
    if request.method == 'POST':
        pass
    else:
        category = Category.objects.all().order_by('id')
        offers = Offer.objects.all()
        context = {
            'categories' :category,
            'offers' : offers
        }
        return render(request, 'admin/category_management.html',context)

# ADD CATEGORY

```

Fig 5.1.9 Category management views.py



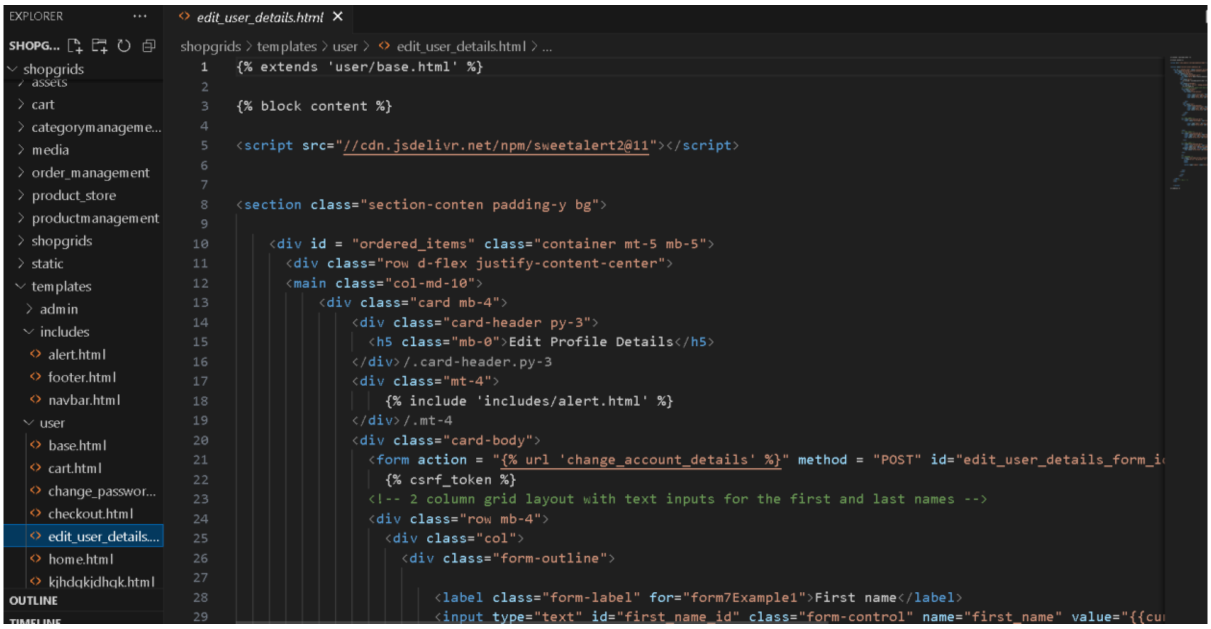
```html

{% if messages %}
    {% for message in messages %}
    <div id="message" class ="container">
        <div {% if message.tags %} class="alert alert-{{ message.tags }}"{% endif %} role= "alert">
            <span arial-hidden="true"></span></button>
            {% if message.level == DEFAULT_MESSAGE_LEVELS_ERROR %}Important: {% endif %}
            {{ message }}

        </div>/.alert.alert-{{.message.tags.}}

    </div>/#message.container
    {% endfor %}

{% endif %}
```

Fig 5.1.10 template alert.html



```html
{% extends 'user/base.html' %}

{% block content %}

<div class="container">
    <div class="row d-flex justify-content-center mt-5 mb-5 p-5 pt-10 pb-10">
        <div class="col-md-5">
            {% include 'includes/alert.html' %}
            <div class="card login-form">
              <div class="card-body">
                <h3 class="card-title text-center">Change password</h3>

                <!--Password must contain one lowercase letter, one number, and be at least 7 characters long.--

                    <div class="card-text  mb-3">
                        <form method="POST" action="{% url 'change_password' %}">
                        {% csrf_token %}

                    </div>/.card-text.mb-3
                    <div class="form-group mb-3">
                      <label for="exampleInputEmail1">Enter Old Password</label>
                      <input type="password" name="old_password" class="form-control form-control-sm">
                    </div>/.form-group.mb-3
                    <div class="form-group mb-3">
                      <label for="exampleInputEmail1">Your new password</label>
                      <input type="password" name="new_password" class="form-control form-control-sm">
                    </div>/.form-group.mb-3
                    <div class="form-group mb-3">
                      <label for="exampleInputEmail1">Repeat password</label>
```

Fig 5.1.11 Changepassword.html



```html
{% extends 'user/base.html' %}

{% block content %}

<script src="//cdn.jsdelivr.net/npm/sweetalert2@11"></script>


<section class="section-conten padding-y bg">

    <div id = "ordered_items" class="container mt-5 mb-5">
      <div class="row d-flex justify-content-center">
      <main class="col-md-10">
          <div class="card mb-4">
              <div class="card-header py-3">
                <h5 class="mb-0">Edit Profile Details</h5>
              </div>/.card-header.py-3
              <div class="mt-4">
                  {% include 'includes/alert.html' %}
              </div>/.mt-4
              <div class="card-body">
                <form action = "{% url 'change_account_details' %}" method = "POST" id="edit_user_details_form_i
                  {% csrf_token %}
                <!-- 2 column grid layout with text inputs for the first and last names -->
                <div class="row mb-4">
                  <div class="col">
                    <div class="form-outline">

                        <label class="form-label" for="form7Example1">First name</label>
                        <input type="text" id="first_name_id" class="form-control" name="first_name" value="{{cu
```

Fig 5.1.12 Userdetails.html

CHAPTER 6

RESULTS AND DISCUSSIONS

Home Page

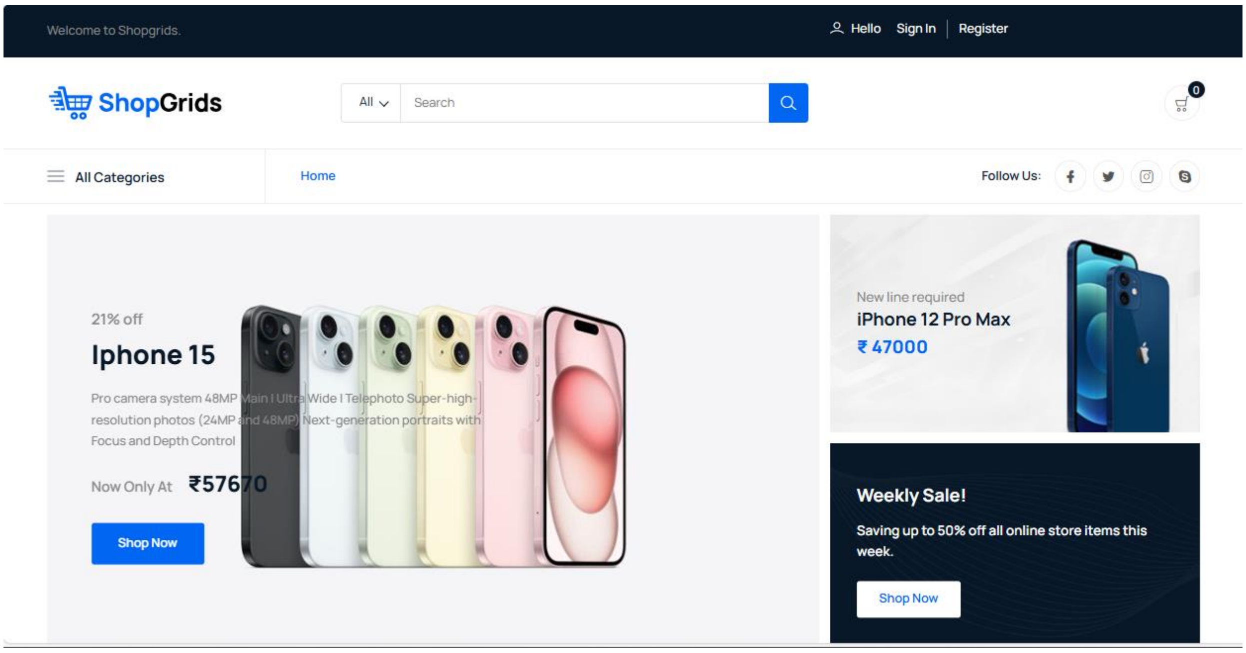
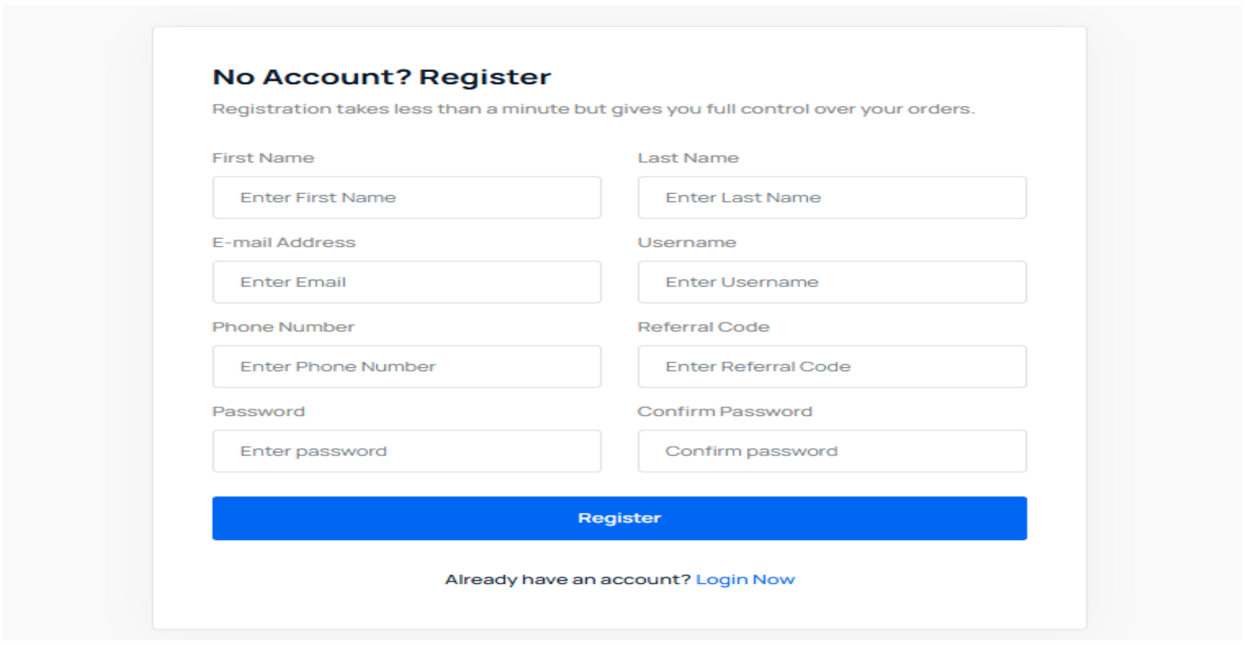

Welcome to Shopgrids.
Hello Sign In | Register
ShopGrids
All
Search
All Categories
Home
Follow Us:
21% off
Iphone 15
Pro camera system 48MP Main I Ultra Wide I Telephoto Super-high-resolution photos (24MP and 48MP) Next-generation portraits with Focus and Depth Control
Now Only At ₹57670
Shop Now
New line required
iPhone 12 Pro Max
₹ 47000
Weekly Sale!
Saving up to 50% off all online store items this week.
Shop Now

Fig 6.1 Home page

Registration Page



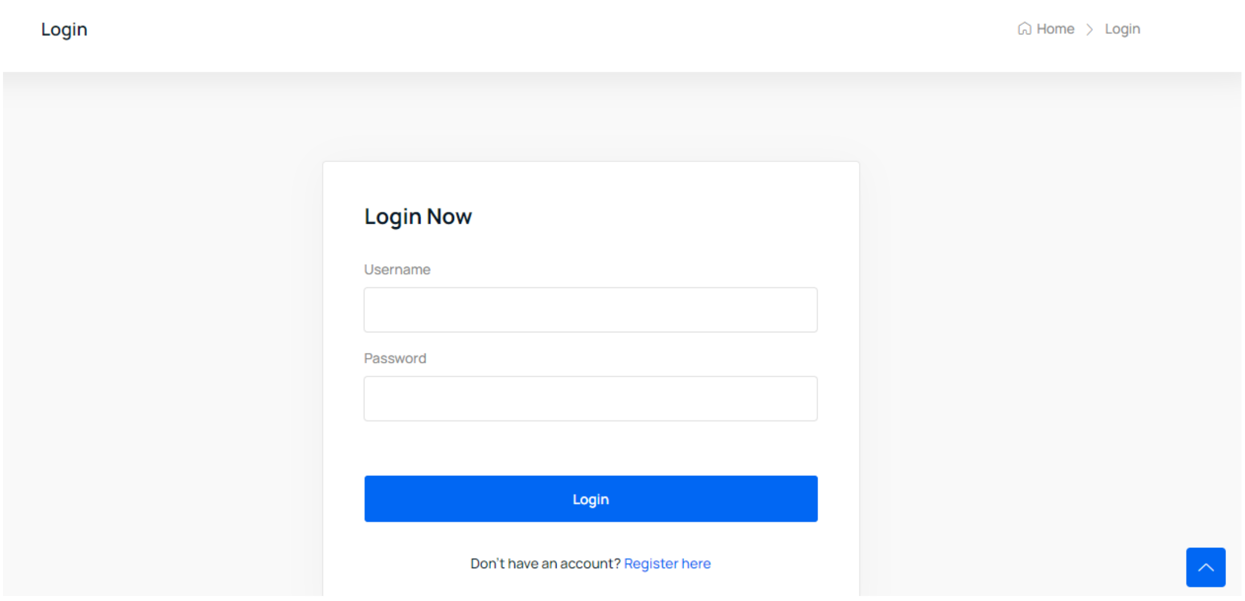
No Account? Register

Registration takes less than a minute but gives you full control over your orders.

First Name: Enter First Name	Last Name: Enter Last Name
E-mail Address: Enter Email	Username: Enter Username
Phone Number: Enter Phone Number	Referral Code: Enter Referral Code
Password: Enter password	Confirm Password: Confirm password

Fig 6.2 Registration page

Login Page



Login

Home > Login

Login Now

Username

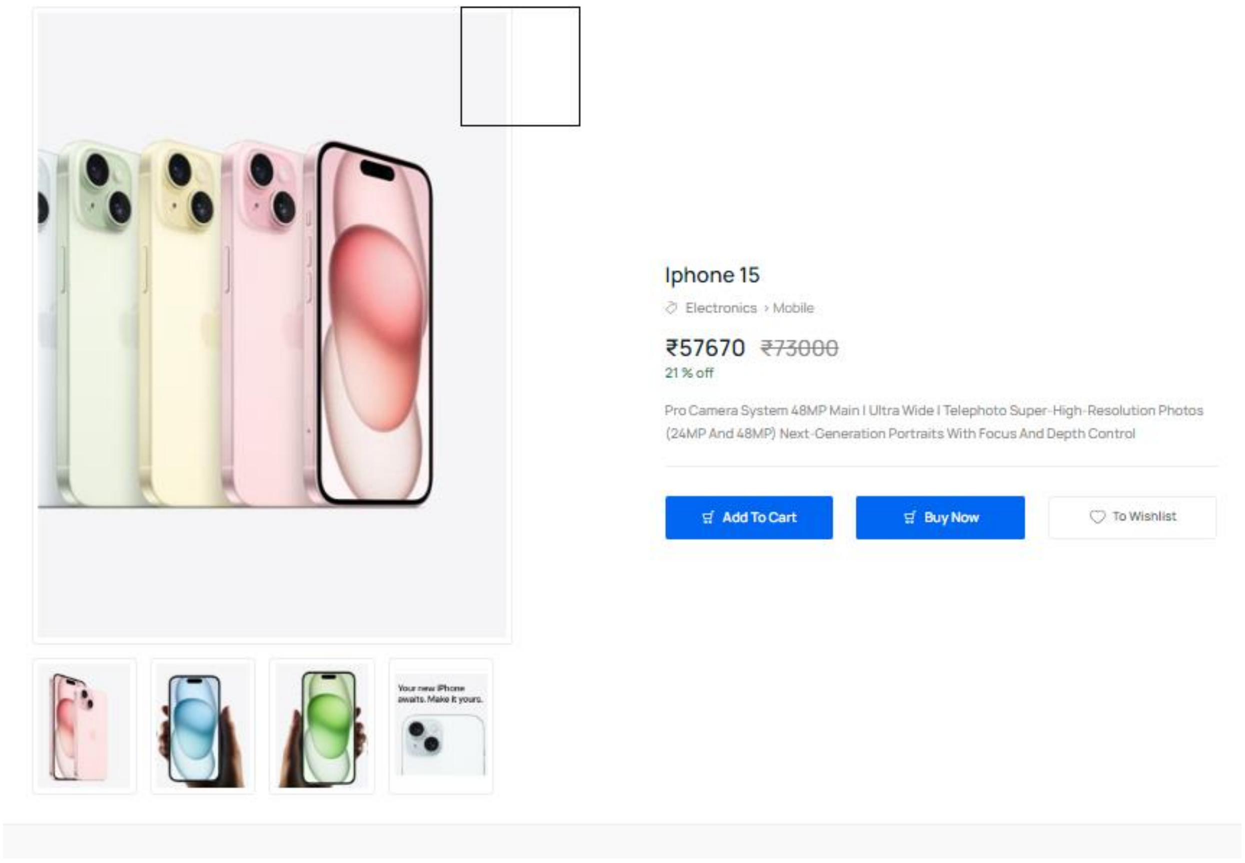
Password

Login

Don't have an account? Register here

Fig 6.3 Login Page

Product Page



Iphone 15

Electronics › Mobile

₹57670 ~~₹73000~~

21 % off

Pro Camera System 48MP Main I Ultra Wide I Telephoto Super-High-Resolution Photos (24MP And 48MP) Next-Generation Portraits With Focus And Depth Control

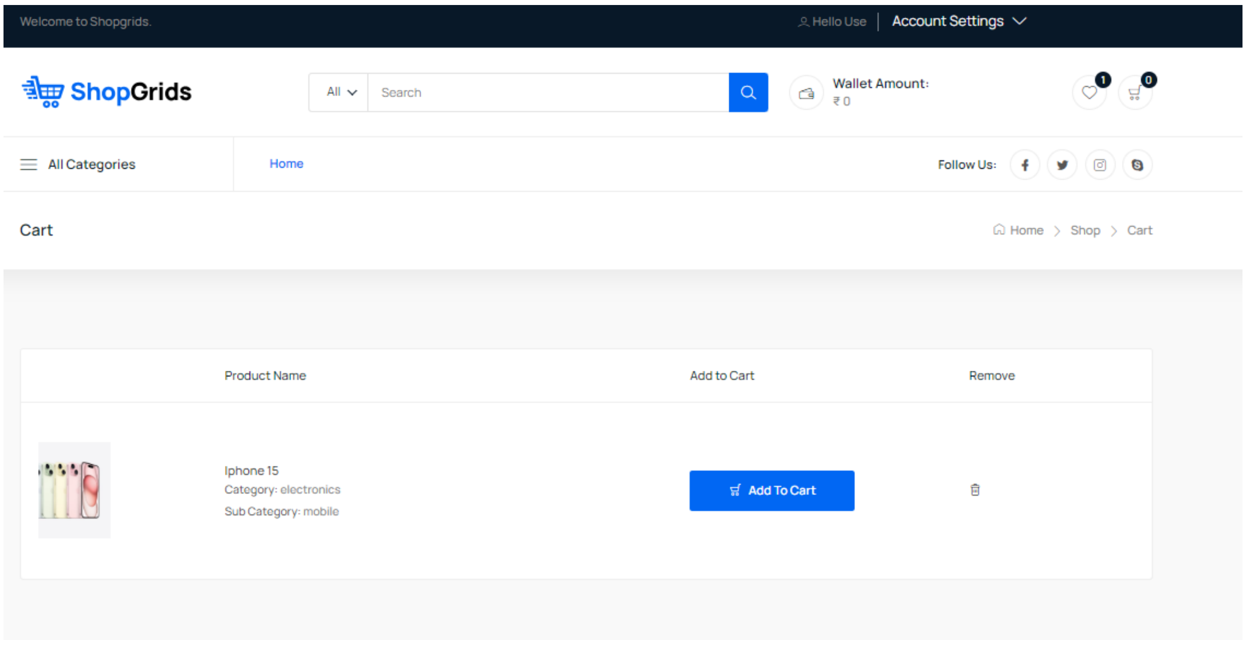
Add To Cart

Buy Now

To Wishlist

Fig 6.4 Product page

Wishlist Page



Welcome to Shopgrids. Hello Use | Account Settings

ShopGrids | All | Search | Wallet Amount: ₹ 0

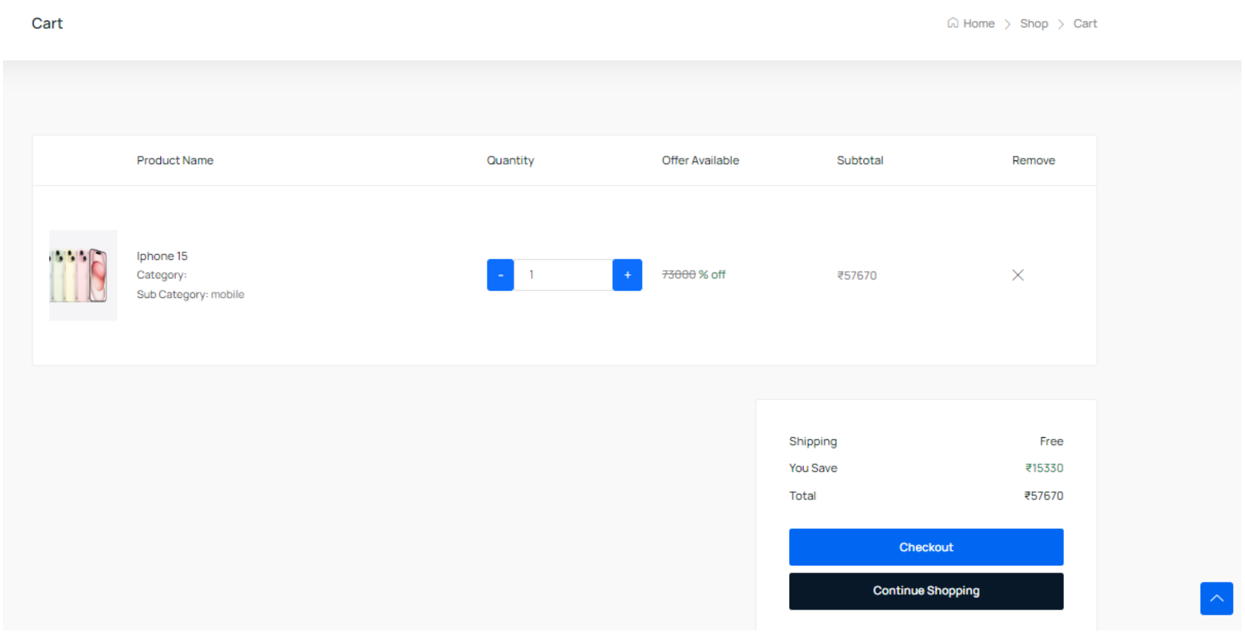
All Categories | Home | Follow Us:

Cart | Home > Shop > Cart

	Product Name	Add to Cart	Remove
	Iphone 15 Category: electronics Sub Category: mobile	Add To Cart	

Fig 6.5 Wishlist Page

Cart Page



Cart | Home > Shop > Cart

	Product Name	Quantity	Offer Available	Subtotal	Remove
	Iphone 15 Category: Sub Category: mobile	- 1 +	~~73000~~ % off	₹57670	

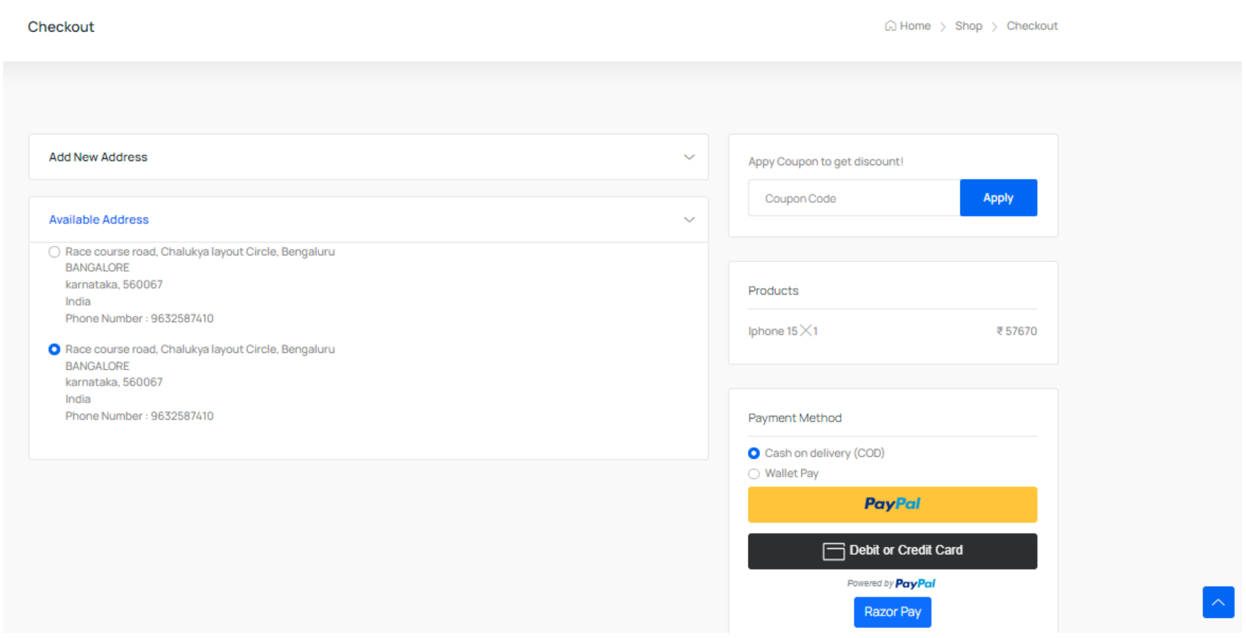
Shipping	Free
You Save	₹15330
Total	₹57670

Checkout

Continue Shopping

Fig 6.6 Cart

Product Checkout Page



Checkout

Home > Shop > Checkout

Add New Address

Available Address

- ○ Race course road, Chalukya layout Circle, Bengaluru
  BANGALORE
  karnataka, 560067
  India
  Phone Number : 9632587410
- ● Race course road, Chalukya layout Circle, Bengaluru
  BANGALORE
  karnataka, 560067
  India
  Phone Number : 9632587410

Appy Coupon to get discount!

Coupon Code | Apply

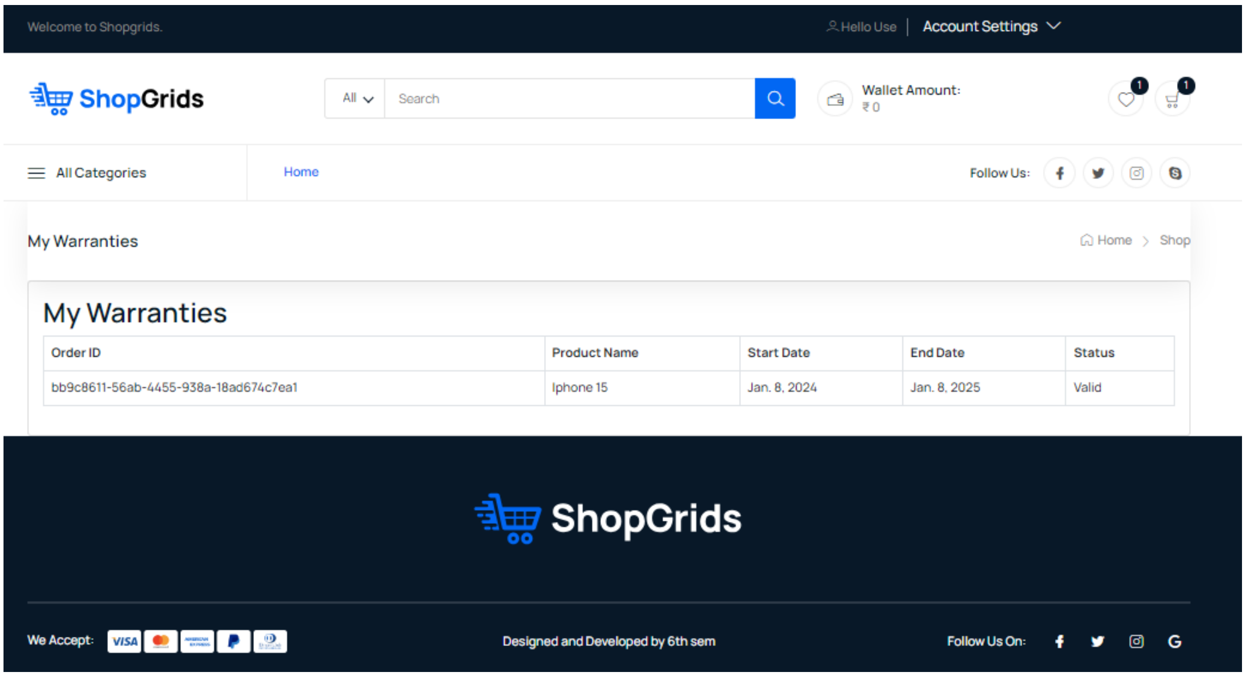
Products	
Iphone 15 ╳ 1	₹ 57670

Payment Method

- ● Cash on delivery (COD)
- ○ Wallet Pay

Fig 6.7 Checkout

View Warranty



My Warranties

Home > Shop

My Warranties

Order ID	Product Name	Start Date	End Date	Status
bb9c8611-56ab-4455-938a-18ad674c7ea1	Iphone 15	Jan. 8, 2024	Jan. 8, 2025	Valid

Fig 6.8 View Warranty

Admin Login Page

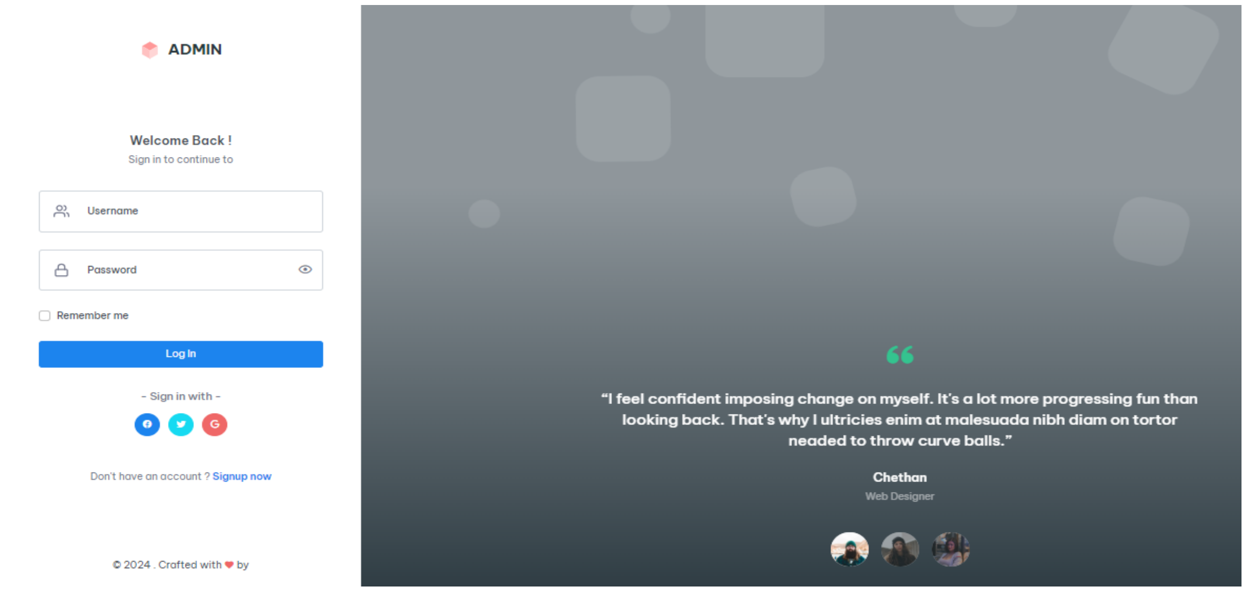

ADMIN
Welcome Back !
Sign in to continue to
Username
Password
Remember me
Log In
- Sign in with -
Don't have an account ? Signup now
© 2024 . Crafted with ❤ by
"I feel confident imposing change on myself. It's a lot more progressing fun than looking back. That's why I ultricies enim at malesuada nibh diam on tortor needed to throw curve balls."
Chethan
Web Designer

Fig 6.9 Admin login page

Admin Dashboard

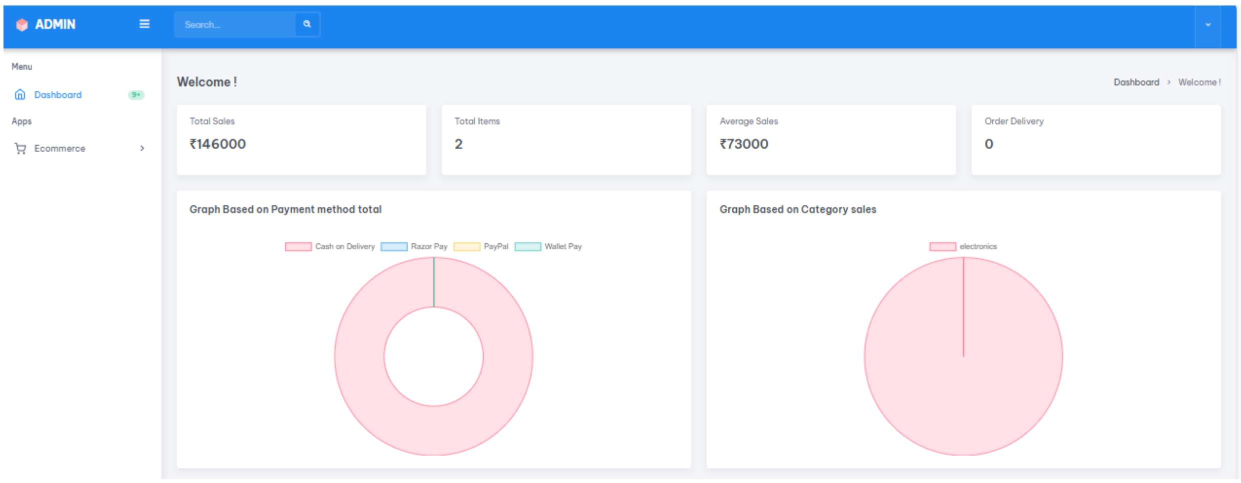
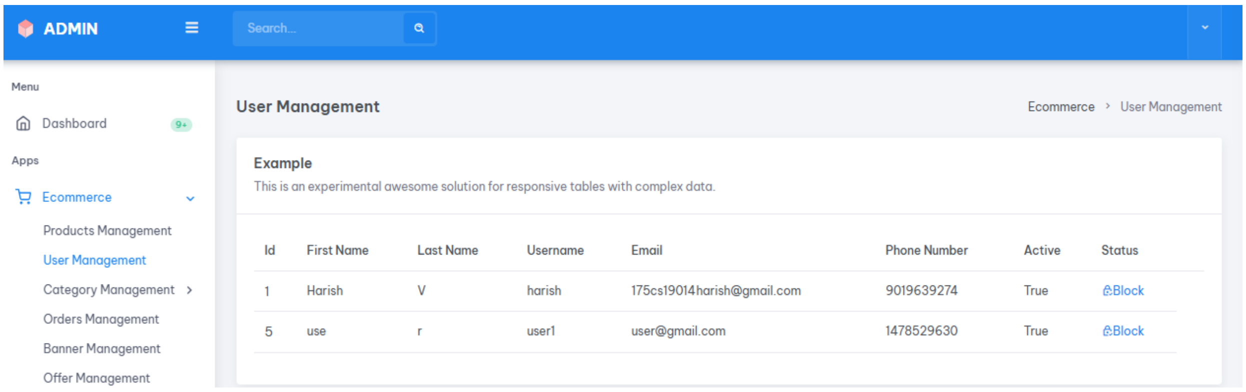


Fig 6.10 Admin dashboard

User Management



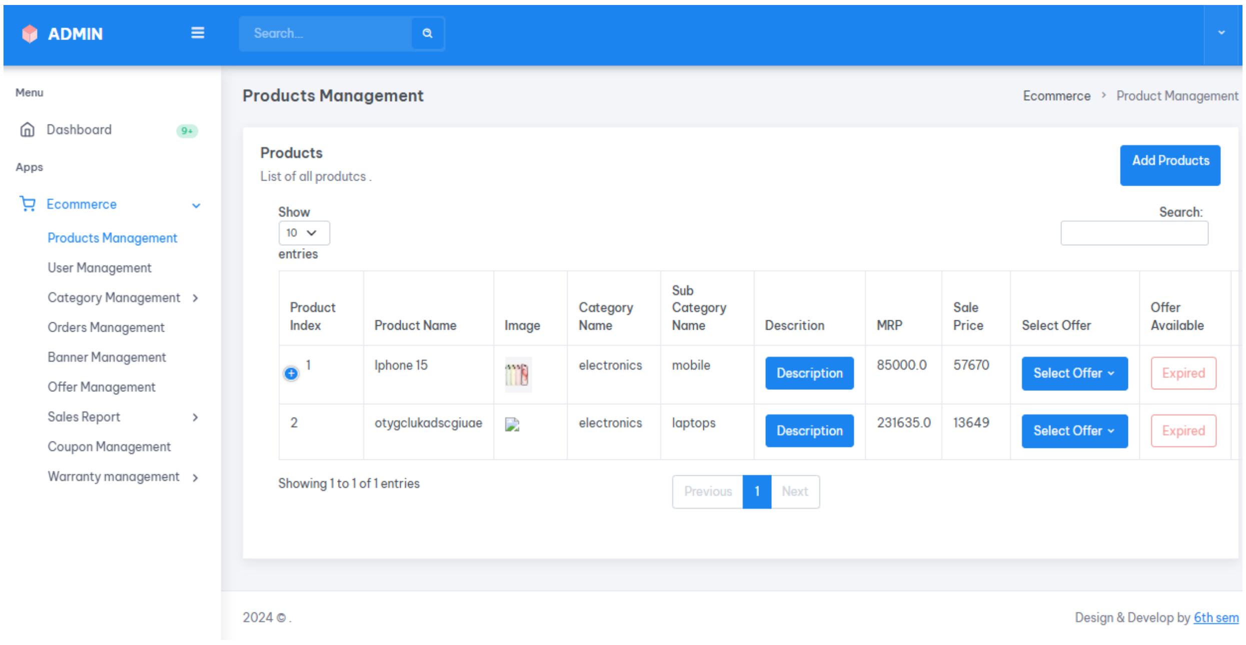
Example

This is an experimental awesome solution for responsive tables with complex data.

Id	First Name	Last Name	Username	Email	Phone Number	Active	Status
1	Harish	V	harish	175cs19014harish@gmail.com	9019639274	True	Block
5	use	r	user1	user@gmail.com	1478529630	True	Block

Fig 6.11 User Management

Product Management



Products Management

Ecommerce › Product Management

Products

List of all produtcs .

Add Products

Show 10 entries

Search:

Product Index	Product Name	Image	Category Name	Sub Category Name	Descrition	MRP	Sale Price	Select Offer	Offer Available
1	Iphone 15		electronics	mobile	Description	85000.0	57670	Select Offer	Expired
2	otygclukadscgiuae		electronics	laptops	Description	231635.0	13649	Select Offer	Expired

Showing 1 to 1 of 1 entries

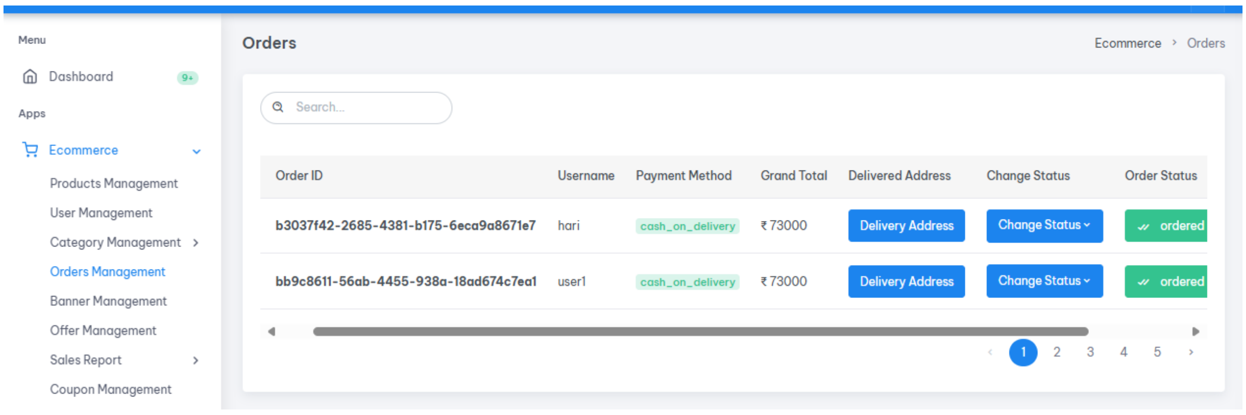
Previous 1 Next

2024 © .

Design & Develop by 6th sem

Fig 6.12 Product Management

Order Management



Orders

Ecommerce › Orders

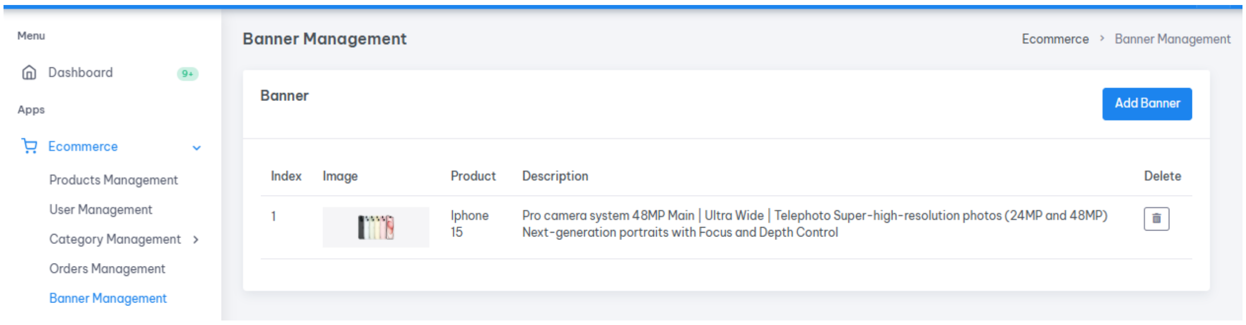
Search...

Order ID	Username	Payment Method	Grand Total	Delivered Address	Change Status	Order Status
b3037f42-2685-4381-b175-6eca9a8671e7	hari	cash_on_delivery	₹ 73000	Delivery Address	Change Status	ordered
bb9c8611-56ab-4455-938a-18ad674c7ea1	user1	cash_on_delivery	₹ 73000	Delivery Address	Change Status	ordered

‹ 1 2 3 4 5 ›

Fig 6.13 Order Management

Banner Management



Banner Management

Ecommerce › Banner Management

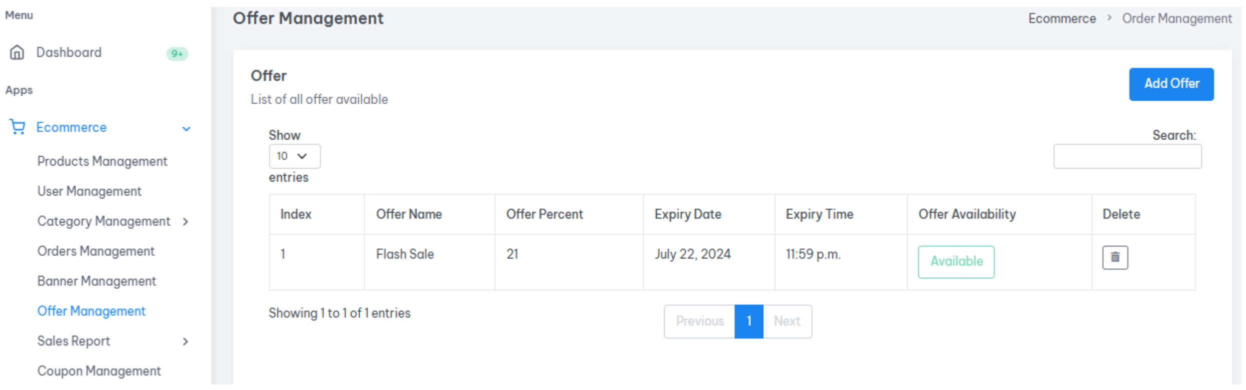
Banner

Add Banner

Index	Image	Product	Description	Delete
1		Iphone 15	Pro camera system 48MP Main | Ultra Wide | Telephoto Super-high-resolution photos (24MP and 48MP) Next-generation portraits with Focus and Depth Control	

Fig 6.14 Banner Management

Offer Management



Offer Management

Ecommerce > Order Management

Offer

List of all offer available

Show 10 entries

Search:

Index	Offer Name	Offer Percent	Expiry Date	Expiry Time	Offer Availability	Delete
1	Flash Sale	21	July 22, 2024	11:59 p.m.	Available	

Showing 1 to 1 of 1 entries

Fig 6.15 Offer Management

CHAPTER 7

CONCLUSION & FUTURE SCOPE

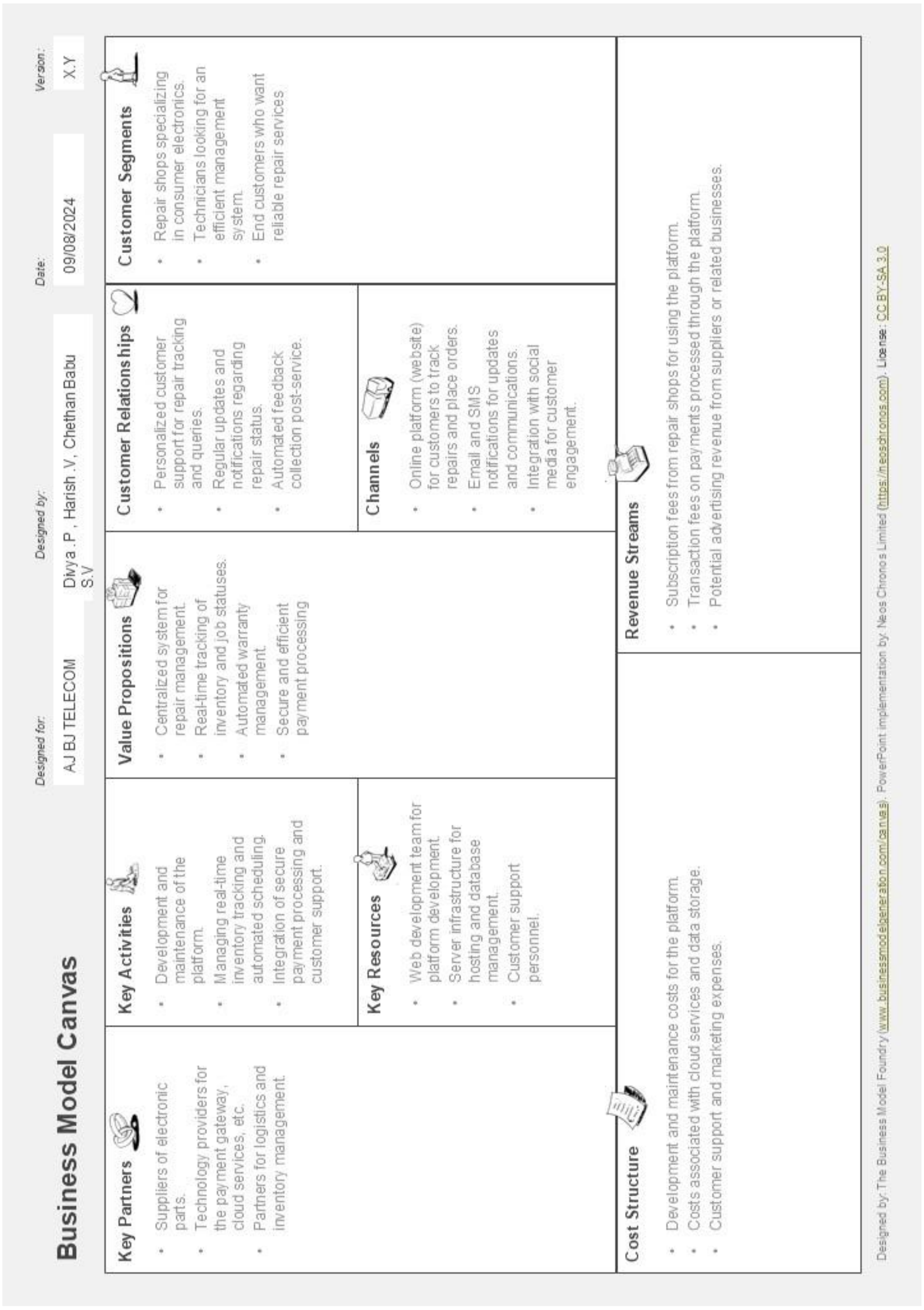
The development of this e-commerce platform for consumer electronics repair management using Django represents a significant advancement in streamlining repair shop operations. By integrating real-time inventory tracking, automated repair scheduling, and robust customer relationship management features, the platform not only enhances operational efficiency but also improves customer satisfaction. The cohesive architecture, with Django serving both the frontend and backend, ensures a seamless and scalable solution that addresses the critical challenges faced by repair professionals in today's fast-paced market.

Looking ahead, the platform has the potential to evolve further by incorporating advanced features such as AI-driven diagnostics and predictive analytics, which could optimize repair processes and inventory management. Additionally, expanding the platform's capabilities to support mobile applications and integrating with IoT devices could offer enhanced accessibility and automation, making the system even more versatile and appealing to a broader range of users in the consumer electronics industry.

Moreover, the platform can be extended to support global operations by implementing multi-language and multi-currency functionalities, catering to a diverse customer base. Integrating third-party services like logistics providers and suppliers could further streamline the end-to-end repair process, offering a more comprehensive solution to repair shops. As the platform continues to evolve, it has the potential to set new standards in the industry, providing a robust, secure, and user-friendly environment that adapts to the ever-changing needs of both repair professionals and consumers.

CHAPTER 8

BUSINESS CANVAS MODEL



Business Model Canvas

Designed for: AJ BJ TELECOM

Designed by: Divya .P , Harish .V, Chethan Babu S.V

Date: 09/08/2024

Version: X.Y

Key Partners
- Suppliers of electronic parts.
- Technology providers for the payment gateway, cloud services, etc.
- Partners for logistics and inventory management.

Key Activities
- Development and maintenance of the platform.
- Managing real-time inventory tracking and automated scheduling.
- Integration of secure payment processing and customer support.

Key Resources
- Web development team for platform development.
- Server infrastructure for hosting and database management.
- Customer support personnel.

Value Propositions
- Centralized system for repair management.
- Real-time tracking of inventory and job statuses.
- Automated warranty management.
- Secure and efficient payment processing

Customer Relationships
- Personalized customer support for repair tracking and queries.
- Regular updates and notifications regarding repair status.
- Automated feedback collection post-service.

Channels
- Online platform (website) for customers to track repairs and place orders.
- Email and SMS notifications for updates and communications.
- Integration with social media for customer engagement.

Customer Segments
- Repair shops specializing in consumer electronics.
- Technicians looking for an efficient management system.
- End customers who want reliable repair services

Cost Structure
- Development and maintenance costs for the platform.
- Costs associated with cloud services and data storage.
- Customer support and marketing expenses.

Revenue Streams
- Subscription fees from repair shops for using the platform.
- Transaction fees on payments processed through the platform.
- Potential advertising revenue from suppliers or related businesses.

Designed by: The Business Model Foundry (www.businessmodelgeneration.com/canvas). PowerPoint implementation by: Neos Chronos Limited (https://neoschronos.com). License: CC BY-SA 3.0

APENDIX

This project leverages a range of modern technologies, including Django for both frontend and backend development, and PostgreSQL/MySQL for robust data management. The user interface is crafted with HTML, CSS, and JavaScript, ensuring a responsive and intuitive design. AJAX is employed to enhance user interaction by enabling asynchronous data exchange without page reloads. A secure payment gateway is integrated for processing transactions, while version control is managed through Git, facilitating collaboration and change management.

Key features of the platform include user registration and authentication, real-time inventory management, repair scheduling and tracking, warranty management, and customer relationship management (CRM). Additionally, it supports billing and payment processing, comprehensive reporting and analytics, and implements strong security measures such as role-based access control and CSRF protection. The development environment is set up on Ubuntu Linux, with Visual Studio Code and PyCharm as primary IDEs. Testing tools like pytest and Selenium are used for automated and end-to-end testing, while Docker, Nginx, and Gunicorn handle deployment.

Assumptions made during development include user familiarity with e-commerce platforms and stable internet connectivity for repair shops. Limitations include potential compatibility issues with older devices and dependency on infrastructure performance. The platform underwent rigorous testing, including unit, integration, and user acceptance testing, as well as security validation to safeguard against vulnerabilities. Deployment follows a structured strategy, with a staging environment for final testing and a CI/CD pipeline ensuring smooth updates.

Looking ahead, the project has the potential to expand through the development of mobile applications for iOS and Android, AI and machine learning integration for enhanced diagnostics and predictive analytics, and globalization features such as multi-language and multi-currency support. Additionally, IoT integration could enable remote monitoring and management of repair processes, further enhancing the platform's capabilities and broadening its appeal to a global audience.

REFERENCES

- Django Software Foundation. (2024). Django Documentation. Retrieved from https://docs.djangoproject.com/en/stable/
- Mozilla Developer Network (MDN). (2024). Ajax Documentation. Retrieved from https://developer.mozilla.org/enUS/docs/Web/Guide/AJAX
- PostgreSQL Global Development Group. (2024). PostgreSQL Documentation. Retrieved from https://www.postgresql.org/docs/
- Oracle Corporation. (2024). MySQL Documentation. Retrieved from https://dev.mysql.com/doc/
- Chacon, S., & Straub, B. (2014). Pro Git. Apress. Retrieved from https://git-scm.com/book/en/v2
- An open-source book on Git, providing in-depth Roy, M. (2024). Full Stack Python. Retrieved from https://www.fullstackpython.com/