

National Public School

**By: Vybhav Aditya, Yashas P, Santhosh S**

Accounting System

Computer Science

Project Report 2023-24



**Acknowledgment**

We extend our heartfelt gratitude to all those who contributed to the realization of this Django Pharmacy Management System. Our deepest appreciation goes to our principal, Mrs. Geeta Dikshit, whose unwavering support and encouragement propelled us forward. Her guidance and vision have been invaluable throughout this endeavor.

Additionally, we express our sincere thanks to Mrs. Uma P, our dedicated computer teacher, whose expertise and guidance have been instrumental in shaping our technical skills. Her continuous support and mentorship have been crucial in steering us through the complexities of software development.

This project stands as a testament to the support and guidance of these remarkable individuals, without whom this achievement would not have been possible. Their encouragement and belief in our abilities have been a constant source of inspiration.

Thank you.

**Index**

|  |  |  |  |
| --- | --- | --- | --- |
| Sl no. | Topic | Page No. | Signature |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
| 16 | Conclusion |  |  |
| 17 | Student reflection |  |  |
| 18 | Bibliography |  |  |
|  | Teacher’s remarks: |  |  |

**Introduction:**

In an era where efficiency, accuracy, and seamless management define the success of any business, the pharmaceutical industry stands at the forefront of innovation. This report delves into the development and implementation of a cutting-edge Django Pharmacy Management System, aimed at revolutionizing the way pharmacies streamline their operations.

In today's dynamic healthcare landscape, the need for robust systems to manage inventory, prescriptions, sales, and customer data is paramount. Recognizing this necessity, our team embarked on a journey to create a comprehensive solution that not only addresses these challenges but also sets new standards for efficiency and precision within the pharmaceutical realm.

This report provides an in-depth analysis of the architecture, functionalities, and impact of the Django-based system in optimizing pharmacy workflows. By leveraging the power of Django, a high-level Python web framework known for its versatility and scalability, this system promises to redefine the operational landscape of pharmacies, providing a cohesive platform to manage diverse facets of the pharmaceutical business seamlessly.

**Objective:**

The primary objectives of this Django Pharmacy Management System project revolve around optimizing operational efficiency, enhancing user experience, and ensuring seamless integration of critical functionalities.

1. Inventory Management: Implementing a robust system to track and manage inventory levels effectively. This includes real-time updates on stock quantities, automated alerts for low-stock items, and streamlined processes for inventory replenishment.
2. Sales and Billing: Creating a reliable platform for sales and billing operations, allowing for smooth transactions, invoice generation, and tracking of sales history. The system aims to streamline payment processing and reduce errors in billing procedures.
3. Customer Management: Establishing a comprehensive database for customer information, enabling efficient customer management, tracking purchase history, and facilitating personalized services based on customer preferences.
4. User-Friendly Interface: Designing an intuitive and easy-to-navigate interface to ensure accessibility for users with varying technical expertise. The system aims to prioritize user experience, enabling efficient operation by staff members.

By focusing on these objectives, the Django Pharmacy Management System aims to revolutionize the way pharmacies operate, enhancing productivity, accuracy, and overall business performance.

**Software Used**

Here's how each software and technology might have been utilized in the project:

1. Python: The primary language for backend development using Django, providing the core logic for the application, including handling database operations, business logic, and API integrations.
2. Django: A high-level Python web framework utilized for rapid development, providing a clean and pragmatic design and facilitating the creation of secure and maintainable web applications.
3. HTML & CSS: Used for structuring the frontend user interface and styling the application to create a visually appealing and user-friendly experience.
4. JavaScript: Employed for client-side scripting to add interactivity, validate forms, and enhance the user interface for a smoother experience.
5. jQuery: A JavaScript library utilized to simplify DOM manipulation, event handling, and AJAX calls, streamlining development for interactive web applications.
6. Ajax (Asynchronous JavaScript and XML): Used to enable asynchronous data retrieval from the server, allowing for seamless updating of content without refreshing the entire page.
7. Bootstrap v5 & Material Design Bootstrap Template: Frameworks and templates utilized to create responsive and mobile-first designs, offering pre-styled components and grids for a consistent and polished UI.
8. Font Awesome: A library of scalable vector icons used to enhance the visual appeal of the application by easily incorporating a variety of icons throughout the interface.

These technologies, frameworks, and libraries were likely integrated and orchestrated to create a cohesive and feature-rich Pharmacy Management System, offering both robust functionality and an intuitive user experience.

**Features and Functionality**

1. Category Management:

- Hierarchical Categorization: Implementing a structured category system allowing for the organization of products into hierarchies (e.g., medication types, over-the-counter, prescription drugs).

- Category Customization: Allowing administrators to create, edit, and delete categories, ensuring flexibility in organizing products according to varying criteria.

2. Product Management:

- Product Information: Enabling the addition, modification, and removal of products with comprehensive details such as name, description, unit price, expiry date, and supplier information.

- Product Variation Management: Handling different variations of products (e.g., different strengths, package sizes) within the same category.

- Product Search and Filtering: Implementing search and filter options for easy retrieval of products based on various parameters like name, category, or supplier.

3. Inventory Management:

- Stock Tracking: Providing real-time visibility into inventory levels, allowing for accurate tracking of available stock and automated alerts for low inventory.

- Stock Replenishment: Automated or manual processes for reordering products when inventory levels reach a specified threshold.

- Inventory Reporting: Generating reports on inventory status, including stock movement, turnover rates, and potential stock-outs.

4. Billing:

- Point-of-Sale (POS) Integration: Facilitating seamless transactions at the point of sale, allowing for quick and accurate billing of purchased items.

- Invoicing: Generating invoices for purchases made by customers, capturing details of products bought, prices, and transaction totals.

- Payment Processing: Integrating various payment methods and recording payment details for completed transactions.

These features collectively streamline the management of a pharmacy's operations, from organizing products into categories, monitoring inventory levels, to facilitating smooth billing processes. The system aims to enhance efficiency, accuracy, and customer service while simplifying the overall management of a pharmacy's inventory and sales processes.

**Code Design**

Certainly! Here's an overview of the features and functionalities of the Django Pharmacy Management System focusing on category management, product management, inventory management, and billing:

1. Category Management:

- Hierarchical Categorization: Implementing a structured category system allowing for the organization of products into hierarchies (e.g., medication types, over-the-counter, prescription drugs).

- Category Customization: Allowing administrators to create, edit, and delete categories, ensuring flexibility in organizing products according to varying criteria.

2. Product Management:

- Product Information: Enabling the addition, modification, and removal of products with comprehensive details such as name, description, unit price, expiry date, and supplier information.

- Product Variation Management: Handling different variations of products (e.g., different strengths, package sizes) within the same category.

- Product Search and Filtering: Implementing search and filter options for easy retrieval of products based on various parameters like name, category, or supplier.

3. Inventory Management:

- Stock Tracking: Providing real-time visibility into inventory levels, allowing for accurate tracking of available stock and automated alerts for low inventory.

- Stock Replenishment: Automated or manual processes for reordering products when inventory levels reach a specified threshold.

- Inventory Reporting: Generating reports on inventory status, including stock movement, turnover rates, and potential stock-outs.

4. Billing:

- Point-of-Sale (POS) Integration: Facilitating seamless transactions at the point of sale, allowing for quick and accurate billing of purchased items.

- Invoicing: Generating invoices for purchases made by customers, capturing details of products bought, prices, and transaction totals.

- Payment Processing: Integrating various payment methods and recording payment details for completed transactions.

These features collectively streamline the management of a pharmacy's operations, from organizing products into categories, monitoring inventory levels, to facilitating smooth billing processes. The system aims to enhance efficiency, accuracy, and customer service while simplifying the overall management of a pharmacy's inventory and sales processes.

**How to run the system?**

1. **Step 1:** Download the project from our Github repo:

**https://github.com/vsabidk/Voting-sys**

**Or use git:**

git clone **https://github.com/vsabidk/pharma**

1. **Step 2:** Set Up Your Environment

* **Install Python**: Ensure Python is installed on your system. You can download and install it from the official Python website.
* **Install requirements:**

pip install -r requirements.txt

1. **Step 3:** Start the project

**Run these commands in the cmd opened the project directory:**

python manage.py makemigrations

python manage.py migrate

python manage.py runserver

**Source Code**

## **views.py**

from email import message

from unicodedata import category

from django.shortcuts import render,redirect

from ims\_django.settings import MEDIA\_ROOT, MEDIA\_URL

import json

from django.contrib import messages

from django.contrib.auth.models import User

from django.http import HttpResponse

from imsApp.forms import SaveStock, UserRegistration, UpdateProfile, UpdatePasswords, SaveCategory, SaveProduct, SaveInvoice, SaveInvoiceItem

from imsApp.models import Category, Product, Stock, Invoice, Invoice\_Item

from cryptography.fernet import Fernet

from django.conf import settings

import base64

context = {

    'page\_title' : 'File Management System',

}

#login

def home(request):

    context['page\_title'] = 'Home'

    context['categories'] = Category.objects.count()

    context['products'] = Product.objects.count()

    context['sales'] = Invoice.objects.count()

    return render(request, 'home.html',context)

# Category

def category\_mgt(request):

    context['page\_title'] = "Product Categories"

    categories = Category.objects.all()

    context['categories'] = categories

    return render(request, 'category\_mgt.html', context)

def save\_category(request):

    resp = {'status':'failed','msg':''}

    if request.method == 'POST':

        if (request.POST['id']).isnumeric():

            category = Category.objects.get(pk=request.POST['id'])

        else:

            category = None

        if category is None:

            form = SaveCategory(request.POST)

        else:

            form = SaveCategory(request.POST, instance= category)

        if form.is\_valid():

            form.save()

            messages.success(request, 'Category has been saved successfully.')

            resp['status'] = 'success'

        else:

            for fields in form:

                for error in fields.errors:

                    resp['msg'] += str(error + "<br>")

    else:

        resp['msg'] = 'No data has been sent.'

    return HttpResponse(json.dumps(resp), content\_type = 'application/json')

def manage\_category(request, pk=None):

    context['page\_title'] = "Manage Category"

    if not pk is None:

        category = Category.objects.get(id = pk)

        context['category'] = category

    else:

        context['category'] = {}

    return render(request, 'manage\_category.html', context)

def delete\_category(request):

    resp = {'status':'failed', 'msg':''}

    if request.method == 'POST':

        try:

            category = Category.objects.get(id = request.POST['id'])

            category.delete()

            messages.success(request, 'Category has been deleted successfully')

            resp['status'] = 'success'

        except Exception as err:

            resp['msg'] = 'Category has failed to delete'

            print(err)

    else:

        resp['msg'] = 'Category has failed to delete'

    return HttpResponse(json.dumps(resp), content\_type="application/json")

# product

def product\_mgt(request):

    context['page\_title'] = "Product List"

    products = Product.objects.all()

    context['products'] = products

    return render(request, 'product\_mgt.html', context)

def save\_product(request):

    resp = {'status':'failed','msg':''}

    if request.method == 'POST':

        if (request.POST['id']).isnumeric():

            product = Product.objects.get(pk=request.POST['id'])

        else:

            product = None

        if product is None:

            form = SaveProduct(request.POST)

        else:

            form = SaveProduct(request.POST, instance= product)

        if form.is\_valid():

            form.save()

            messages.success(request, 'Product has been saved successfully.')

            resp['status'] = 'success'

        else:

            for fields in form:

                for error in fields.errors:

                    resp['msg'] += str(error + "<br>")

    else:

        resp['msg'] = 'No data has been sent.'

    return HttpResponse(json.dumps(resp), content\_type = 'application/json')

def manage\_product(request, pk=None):

    context['page\_title'] = "Manage Product"

    if not pk is None:

        product = Product.objects.get(id = pk)

        context['product'] = product

    else:

        context['product'] = {}

    return render(request, 'manage\_product.html', context)

def delete\_product(request):

    resp = {'status':'failed', 'msg':''}

    if request.method == 'POST':

        try:

            product = Product.objects.get(id = request.POST['id'])

            product.delete()

            messages.success(request, 'Product has been deleted successfully')

            resp['status'] = 'success'

        except Exception as err:

            resp['msg'] = 'Product has failed to delete'

            print(err)

    else:

        resp['msg'] = 'Product has failed to delete'

    return HttpResponse(json.dumps(resp), content\_type="application/json")

#Inventory

def inventory(request):

    context['page\_title'] = 'Inventory'

    products = Product.objects.all()

    context['products'] = products

    return render(request, 'inventory.html', context)

#Inventory History

def inv\_history(request, pk= None):

    context['page\_title'] = 'Inventory History'

    if pk is None:

        messages.error(request, "Product ID is not recognized")

        return redirect('inventory-page')

    else:

        product = Product.objects.get(id = pk)

        stocks = Stock.objects.filter(product = product).all()

        context['product'] = product

        context['stocks'] = stocks

        return render(request, 'inventory-history.html', context )

#Stock Form

def manage\_stock(request,pid = None ,pk = None):

    if pid is None:

        messages.error(request, "Product ID is not recognized")

        return redirect('inventory-page')

    context['pid'] = pid

    if pk is None:

        context['page\_title'] = "Add New Stock"

        context['stock'] = {}

    else:

        context['page\_title'] = "Manage New Stock"

        stock = Stock.objects.get(id = pk)

        context['stock'] = stock

    return render(request, 'manage\_stock.html', context )

def save\_stock(request):

    resp = {'status':'failed','msg':''}

    if request.method == 'POST':

        if (request.POST['id']).isnumeric():

            stock = Stock.objects.get(pk=request.POST['id'])

        else:

            stock = None

        if stock is None:

            form = SaveStock(request.POST)

        else:

            form = SaveStock(request.POST, instance= stock)

        if form.is\_valid():

            form.save()

            messages.success(request, 'Stock has been saved successfully.')

            resp['status'] = 'success'

        else:

            for fields in form:

                for error in fields.errors:

                    resp['msg'] += str(error + "<br>")

    else:

        resp['msg'] = 'No data has been sent.'

    return HttpResponse(json.dumps(resp), content\_type = 'application/json')

def delete\_stock(request):

    resp = {'status':'failed', 'msg':''}

    if request.method == 'POST':

        try:

            stock = Stock.objects.get(id = request.POST['id'])

            stock.delete()

            messages.success(request, 'Stock has been deleted successfully')

            resp['status'] = 'success'

        except Exception as err:

            resp['msg'] = 'Stock has failed to delete'

            print(err)

    else:

        resp['msg'] = 'Stock has failed to delete'

    return HttpResponse(json.dumps(resp), content\_type="application/json")

def sales\_mgt(request):

    context['page\_title'] = 'Sales'

    products = Product.objects.filter(status = 1).all()

    context['products'] = products

    return render(request,'sales.html', context)

def get\_product(request,pk = None):

    resp = {'status':'failed','data':{},'msg':''}

    if pk is None:

        resp['msg'] = 'Product ID is not recognized'

    else:

        product = Product.objects.get(id = pk)

        resp['data']['product'] = str(product.code + " - " + product.name)

        resp['data']['id'] = product.id

        resp['data']['price'] = product.price

        resp['status'] = 'success'

    return HttpResponse(json.dumps(resp),content\_type="application/json")

def save\_sales(request):

    resp = {'status':'failed', 'msg' : ''}

    id = 2

    if request.method == 'POST':

        pids = request.POST.getlist('pid[]')

        invoice\_form = SaveInvoice(request.POST)

        if invoice\_form.is\_valid():

            invoice\_form.save()

            invoice = Invoice.objects.last()

            for pid in pids:

                data = {

                    'invoice':invoice.id,

                    'product':pid,

                    'quantity':request.POST['quantity['+str(pid)+']'],

                    'price':request.POST['price['+str(pid)+']'],

                }

                print(data)

                ii\_form = SaveInvoiceItem(data=data)

                # print(ii\_form.data)

                if ii\_form.is\_valid():

                    ii\_form.save()

                else:

                    for fields in ii\_form:

                        for error in fields.errors:

                            resp['msg'] += str(error + "<br>")

                    break

            messages.success(request, "Sale Transaction has been saved.")

            resp['status'] = 'success'

            # invoice.delete()

        else:

            for fields in invoice\_form:

                for error in fields.errors:

                    resp['msg'] += str(error + "<br>")

    return HttpResponse(json.dumps(resp),content\_type="application/json")

def invoices(request):

    invoice =  Invoice.objects.all()

    context['page\_title'] = 'Invoices'

    context['invoices'] = invoice

    return render(request, 'invoices.html', context)

def delete\_invoice(request):

    resp = {'status':'failed', 'msg':''}

    if request.method == 'POST':

        try:

            invoice = Invoice.objects.get(id = request.POST['id'])

            invoice.delete()

            messages.success(request, 'Invoice has been deleted successfully')

            resp['status'] = 'success'

        except Exception as err:

            resp['msg'] = 'Invoice has failed to delete'

            print(err)

    else:

        resp['msg'] = 'Invoice has failed to delete'

    return HttpResponse(json.dumps(resp), content\_type="application/json")

## **models.py**

from re import I

from django.db import models

from django.utils import timezone

from django.dispatch import receiver

from more\_itertools import quantify

from django.db.models import Sum

# Create your models here.

class Category(models.Model):

    name = models.CharField(max\_length=250)

    description = models.TextField()

    status = models.CharField(max\_length=2, choices=(('1','Active'),('2','Inactive')), default=1)

    date\_created = models.DateTimeField(default=timezone.now)

    date\_updated = models.DateTimeField(auto\_now=True)

    def \_\_str\_\_(self):

        return self.name

class Product(models.Model):

    code=models.CharField(max\_length=100,blank=True, null=True)

    name=models.CharField(max\_length=250,blank=True, null=True)

    description = models.TextField()

    price = models.FloatField(default=0)

    status = models.CharField(max\_length=2, choices=(('1','Active'),('2','Inactive')), default=1)

    date\_created = models.DateTimeField(default=timezone.now)

    date\_updated = models.DateTimeField(auto\_now=True)

    def \_\_str\_\_(self):

        return self.code + ' - ' + self.name

    def count\_inventory(self):

        stocks = Stock.objects.filter(product = self)

        stockIn = 0

        stockOut = 0

        for stock in stocks:

            if stock.type == '1':

                stockIn = int(stockIn) + int(stock.quantity)

            else:

                stockOut = int(stockOut) + int(stock.quantity)

        available  = stockIn - stockOut

        return available

class Stock(models.Model):

    product = models.ForeignKey(Product, on\_delete=models.CASCADE)

    quantity = models.FloatField(default=0)

    type = models.CharField(max\_length=2,choices=(('1','Stock-in'),('2','Stock-Out')), default = 1)

    date\_created = models.DateTimeField(default=timezone.now)

    date\_updated = models.DateTimeField(auto\_now=True)

    def \_\_str\_\_(self):

        return self.product.code + ' - ' + self.product.name

class Invoice(models.Model):

    transaction = models.CharField(max\_length = 250)

    customer = models.CharField(max\_length = 250)

    total = models.FloatField(default= 0)

    date\_created = models.DateTimeField(default=timezone.now)

    date\_updated = models.DateTimeField(auto\_now=True)

    def \_\_str\_\_(self):

        return self.transaction

    def item\_count(self):

        return Invoice\_Item.objects.filter(invoice = self).aggregate(Sum('quantity'))['quantity\_\_sum']

class Invoice\_Item(models.Model):

    invoice = models.ForeignKey(Invoice, on\_delete=models.CASCADE)

    product = models.ForeignKey(Product, on\_delete=models.CASCADE)

    stock = models.ForeignKey(Stock, on\_delete=models.CASCADE, blank= True, null= True)

    price = models.FloatField(default=0)

    quantity = models.FloatField(default=0)

    def \_\_str\_\_(self):

        return self.invoice.transaction

@receiver(models.signals.post\_save, sender=Invoice\_Item)

def stock\_update(sender, instance, \*\*kwargs):

    stock = Stock(product = instance.product, quantity = instance.quantity, type = 2)

    stock.save()

    # stockID = Stock.objects.last().id

    Invoice\_Item.objects.filter(id= instance.id).update(stock=stock)

@receiver(models.signals.post\_delete, sender=Invoice\_Item)

def delete\_stock(sender, instance, \*\*kwargs):

    try:

        stock = Stock.objects.get(id=instance.stock.id).delete()

    except:

        return instance.stock.id

## **forms.py**

from django import forms

from django.contrib.auth.forms import UserCreationForm,PasswordChangeForm, UserChangeForm

from django.contrib.auth.models import User

from more\_itertools import quantify

from .models import Category, Product, Stock, Invoice, Invoice\_Item

from datetime import datetime

class UserRegistration(UserCreationForm):

    email = forms.EmailField(max\_length=250,help\_text="The email field is required.")

    first\_name = forms.CharField(max\_length=250,help\_text="The First Name field is required.")

    last\_name = forms.CharField(max\_length=250,help\_text="The Last Name field is required.")

    class Meta:

        model = User

        fields = ('email', 'username', 'password1', 'password2', 'first\_name', 'last\_name')

    def clean\_email(self):

        email = self.cleaned\_data['email']

        try:

            user = User.objects.get(email = email)

        except Exception as e:

            return email

        raise forms.ValidationError(f"The {user.email} mail is already exists/taken")

    def clean\_username(self):

        username = self.cleaned\_data['username']

        try:

            user = User.objects.get(username = username)

        except Exception as e:

            return username

        raise forms.ValidationError(f"The {user.username} mail is already exists/taken")

class UpdateProfile(UserChangeForm):

    username = forms.CharField(max\_length=250,help\_text="The Username field is required.")

    email = forms.EmailField(max\_length=250,help\_text="The Email field is required.")

    first\_name = forms.CharField(max\_length=250,help\_text="The First Name field is required.")

    last\_name = forms.CharField(max\_length=250,help\_text="The Last Name field is required.")

    current\_password = forms.CharField(max\_length=250)

    class Meta:

        model = User

        fields = ('email', 'username','first\_name', 'last\_name')

    def clean\_current\_password(self):

        if not self.instance.check\_password(self.cleaned\_data['current\_password']):

            raise forms.ValidationError(f"Password is Incorrect")

    def clean\_email(self):

        email = self.cleaned\_data['email']

        try:

            user = User.objects.exclude(id=self.cleaned\_data['id']).get(email = email)

        except Exception as e:

            return email

        raise forms.ValidationError(f"The {user.email} mail is already exists/taken")

    def clean\_username(self):

        username = self.cleaned\_data['username']

        try:

            user = User.objects.exclude(id=self.cleaned\_data['id']).get(username = username)

        except Exception as e:

            return username

        raise forms.ValidationError(f"The {user.username} mail is already exists/taken")

class UpdatePasswords(PasswordChangeForm):

    old\_password = forms.CharField(widget=forms.PasswordInput(attrs={'class':'form-control form-control-sm rounded-0'}), label="Old Password")

    new\_password1 = forms.CharField(widget=forms.PasswordInput(attrs={'class':'form-control form-control-sm rounded-0'}), label="New Password")

    new\_password2 = forms.CharField(widget=forms.PasswordInput(attrs={'class':'form-control form-control-sm rounded-0'}), label="Confirm New Password")

    class Meta:

        model = User

        fields = ('old\_password','new\_password1', 'new\_password2')

class SaveCategory(forms.ModelForm):

    name = forms.CharField(max\_length="250")

    description = forms.Textarea()

    status = forms.ChoiceField(choices=[('1','Active'),('2','Inactive')])

    class Meta:

        model = Category

        fields = ('name','description','status')

    def clean\_name(self):

        id = self.instance.id if self.instance.id else 0

        name = self.cleaned\_data['name']

        # print(int(id) > 0)

        # raise forms.ValidationError(f"{name} Category Already Exists.")

        try:

            if int(id) > 0:

                category = Category.objects.exclude(id=id).get(name = name)

            else:

                category = Category.objects.get(name = name)

        except:

            return name

            # raise forms.ValidationError(f"{name} Category Already Exists.")

        raise forms.ValidationError(f"{name} Category Already Exists.")

class SaveProduct(forms.ModelForm):

    name = forms.CharField(max\_length="250")

    description = forms.Textarea()

    status = forms.ChoiceField(choices=[('1','Active'),('2','Inactive')])

    description = forms.CharField(max\_length=250)

    class Meta:

        model = Product

        fields = ('code','name','description','status','price')

    def clean\_code(self):

        id = self.instance.id if self.instance.id else 0

        code = self.cleaned\_data['code']

        try:

            if int(id) > 0:

                product = Product.objects.exclude(id=id).get(code = code)

            else:

                product = Product.objects.get(code = code)

        except:

            return code

        raise forms.ValidationError(f"{code} Category Already Exists.")

class SaveStock(forms.ModelForm):

    product = forms.CharField(max\_length=30)

    quantity = forms.CharField(max\_length=250)

    type = forms.ChoiceField(choices=[('1','Stock-in'),('2','Stock-Out')])

    class Meta:

        model = Stock

        fields = ('product', 'quantity', 'type')

    def clean\_product(self):

        pid = self.cleaned\_data['product']

        try:

            product = Product.objects.get(id=pid)

            print(product)

            return product

        except:

            raise forms.ValidationError("Product is not valid")

class SaveInvoice(forms.ModelForm):

    transaction = forms.CharField(max\_length=100)

    customer = forms.CharField(max\_length=250)

    total = forms.FloatField()

    class Meta:

        model = Invoice

        fields = ('transaction', 'customer', 'total')

    def clean\_transaction(self):

        pref = datetime.today().strftime('%Y%m%d')

        transaction= ''

        code = str(1).zfill(4)

        while True:

            invoice = Invoice.objects.filter(transaction=str(pref + code)).count()

            if invoice > 0:

                code = str(int(code) + 1).zfill(4)

            else:

                transaction = str(pref + code)

                break

        return transaction

class SaveInvoiceItem(forms.ModelForm):

    invoice = forms.CharField(max\_length=30)

    product = forms.CharField(max\_length=30)

    quantity = forms.CharField(max\_length=100)

    price = forms.CharField(max\_length=100)

    class Meta:

        model = Invoice\_Item

        fields = ('invoice','product','quantity','price')

    def clean\_invoice(self):

        iid = self.cleaned\_data['invoice']

        try:

            invoice = Invoice.objects.get(id=iid)

            return invoice

        except:

            raise forms.ValidationError("Invoice ID is not valid")

    def clean\_product(self):

        pid = self.cleaned\_data['product']

        try:

            product = Product.objects.get(id=pid)

            return product

        except:

            raise forms.ValidationError("Product is not valid")

    def clean\_quantity(self):

        qty = self.cleaned\_data['quantity']

        if qty.isnumeric():

            return int(qty)

        raise forms.ValidationError("Quantity is not valid")

## **urls.py**

from django.contrib import admin

from django.urls import path

from django.conf.urls.static import static

from . import views

from django.contrib.auth import views as auth\_views

from django.views.generic.base import RedirectView

urlpatterns = [

    path('redirect-admin', RedirectView.as\_view(url="/admin"),name="redirect-admin"),

    path('',views.home,name='home-page'),

    path('category',views.category\_mgt,name='category-page'),

    path('manage\_category',views.manage\_category,name='manage-category'),

    path('save\_category',views.save\_category,name='save-category'),

    path('manage\_category/<int:pk>',views.manage\_category,name='manage-category-pk'),

    path('delete\_category',views.delete\_category,name='delete-category'),

    path('product',views.product\_mgt,name='product-page'),

    path('manage\_product',views.manage\_product,name='manage-product'),

    path('save\_product',views.save\_product,name='save-product'),

    path('manage\_product/<int:pk>',views.manage\_product,name='manage-product-pk'),

    path('delete\_product',views.delete\_product,name='delete-product'),

    path('inventory',views.inventory,name='inventory-page'),

    path('inventory/<int:pk>',views.inv\_history,name='inventory-history-page'),

    path('stock/<int:pid>',views.manage\_stock,name='manage-stock'),

    path('stock/<int:pid>/<int:pk>',views.manage\_stock,name='manage-stock-pk'),

    path('save\_stock',views.save\_stock,name='save-stock'),

    path('delete\_stock',views.delete\_stock,name='delete-stock'),

    path('sales',views.sales\_mgt,name='sales-page'),

    path('get\_product',views.get\_product,name='get-product'),

    path('get\_product/<int:pk>',views.get\_product),

    path('save\_sales',views.save\_sales, name="save-sales"),

    path('invoices',views.invoices,name='invoice-page'),

    path('delete\_invoice',views.delete\_invoice,name='delete-invoice'),

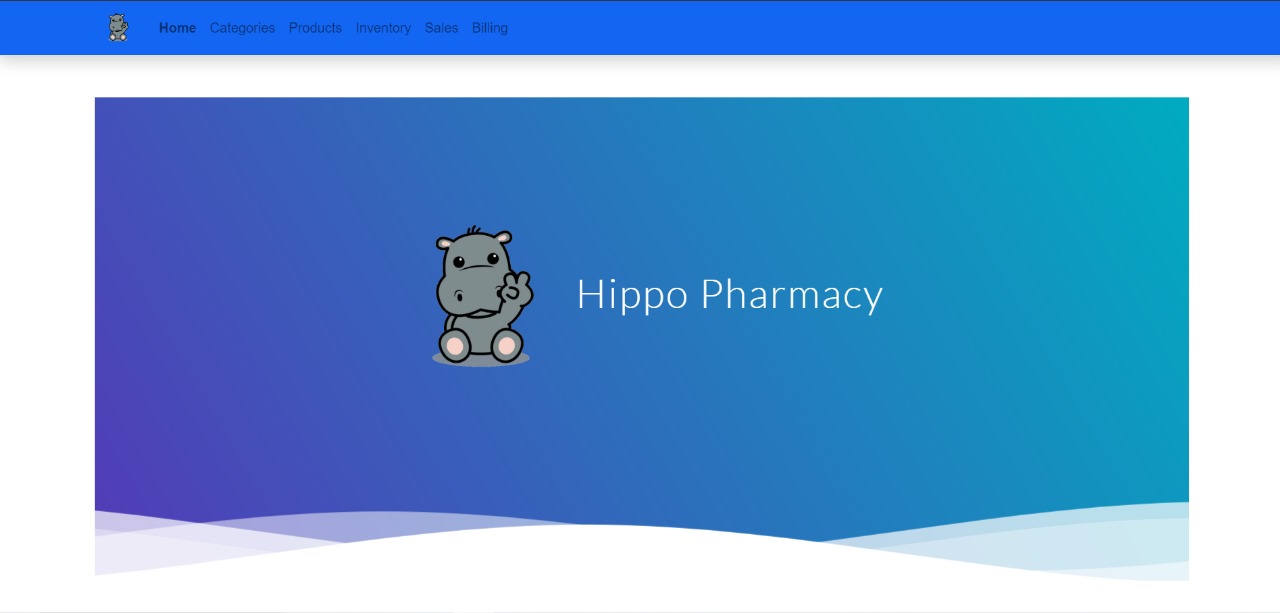
]

**The full source code for our project can be accessed here:**

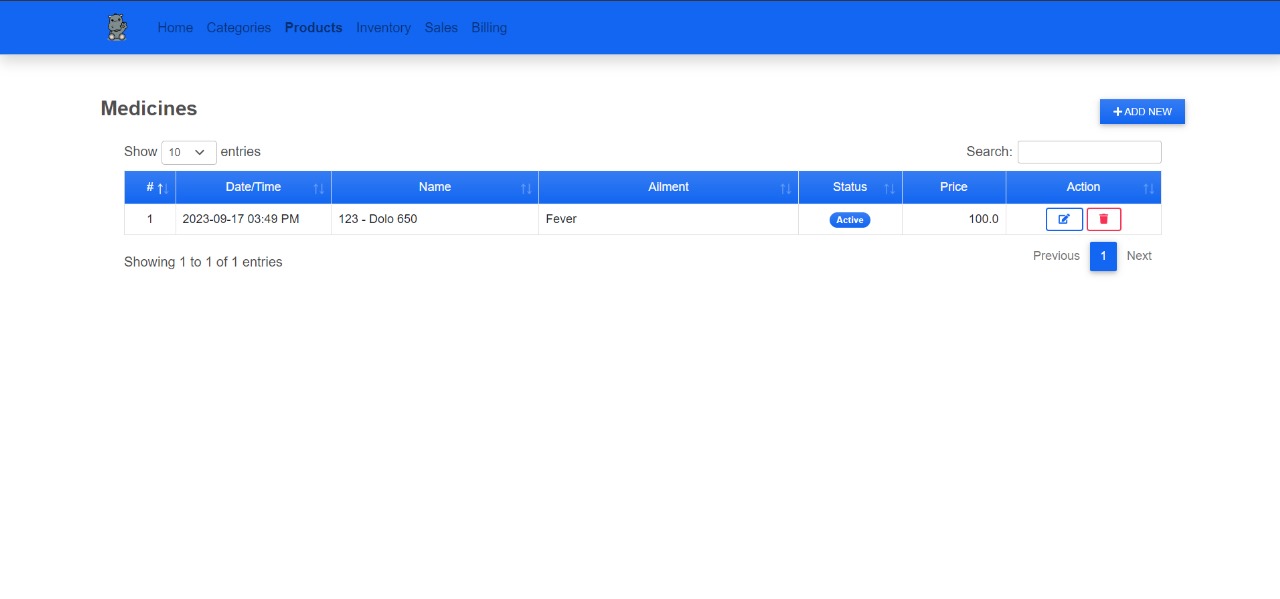
[**https://github.com/vsabidk/pharma**](https://github.com/vsabidk/pharma)

**Output**

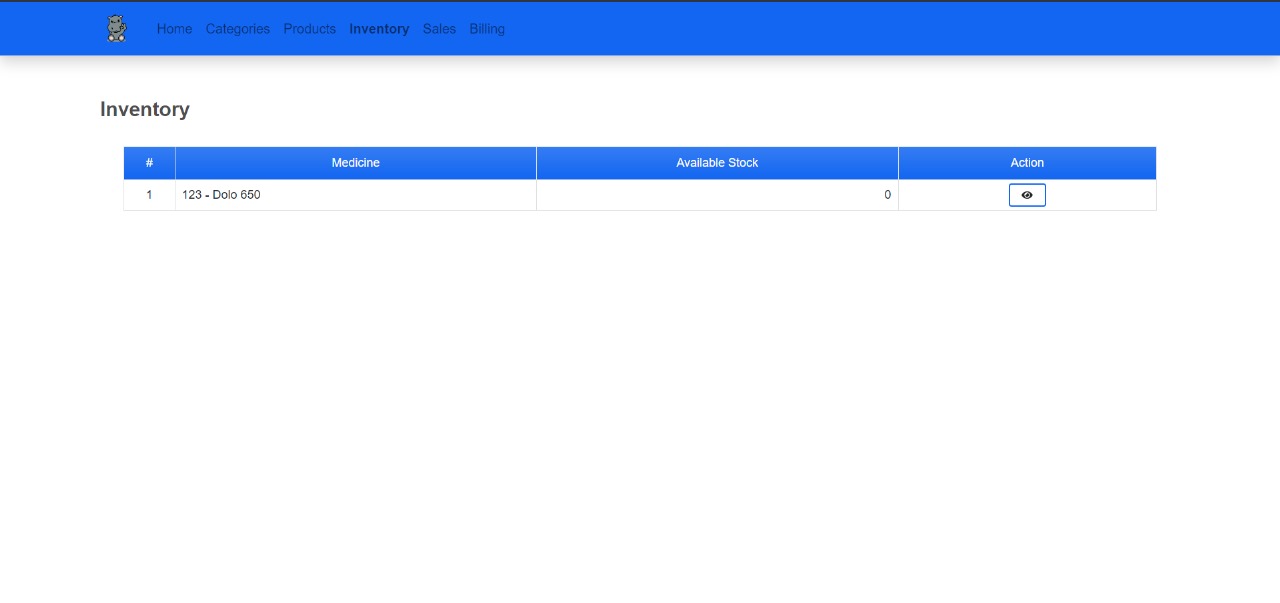
**Home Page:**



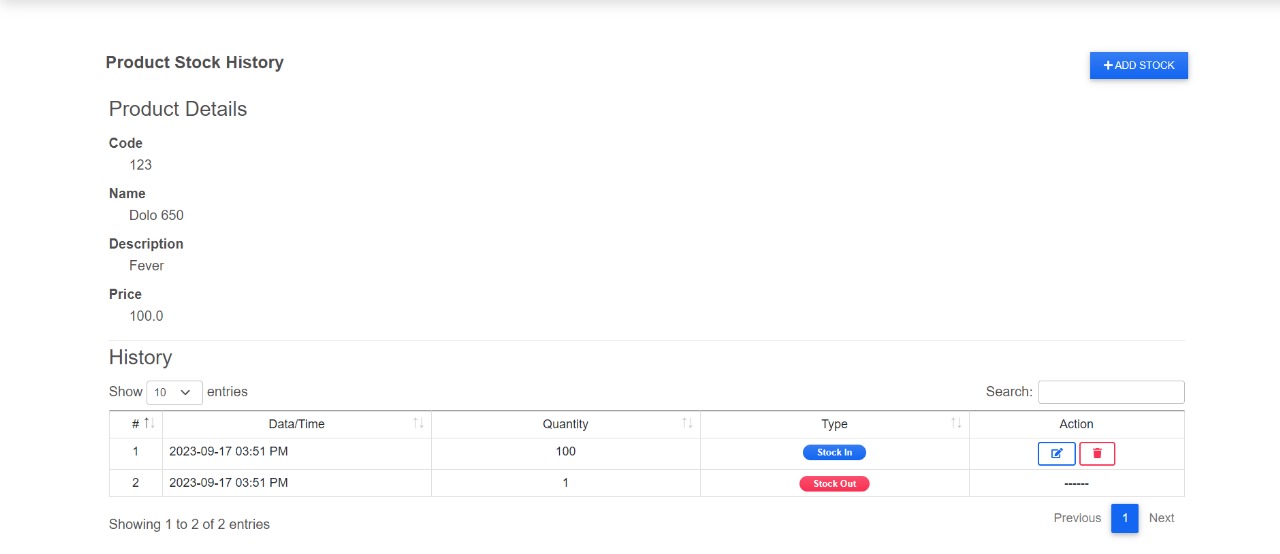
**Product Page**



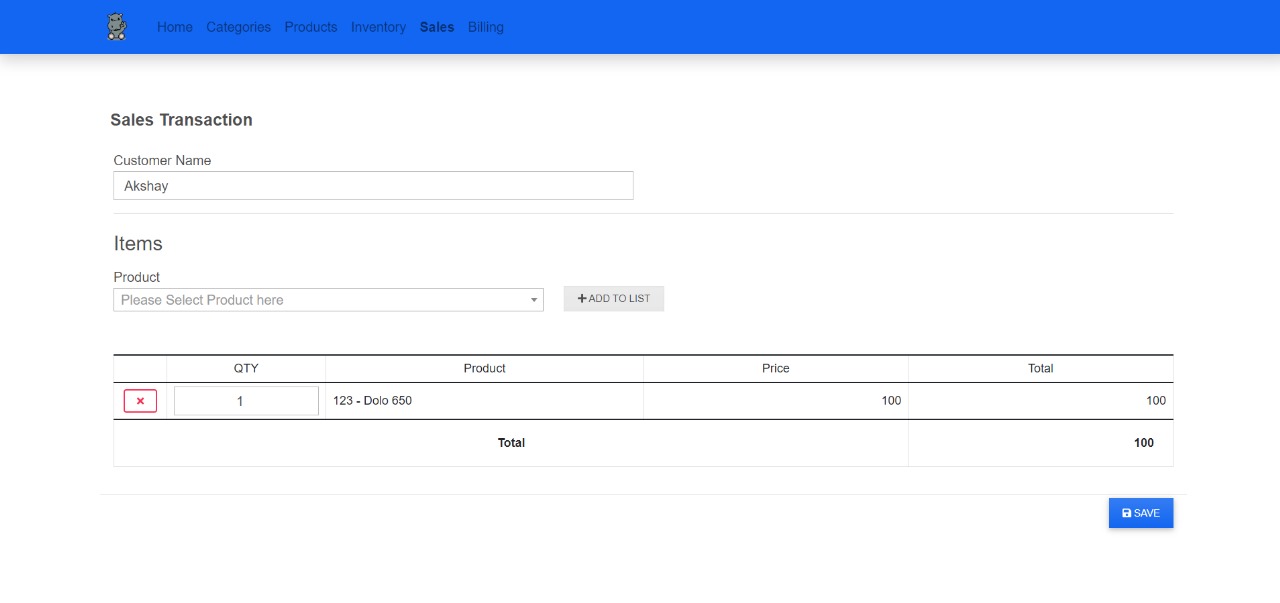
**Inventory Page**



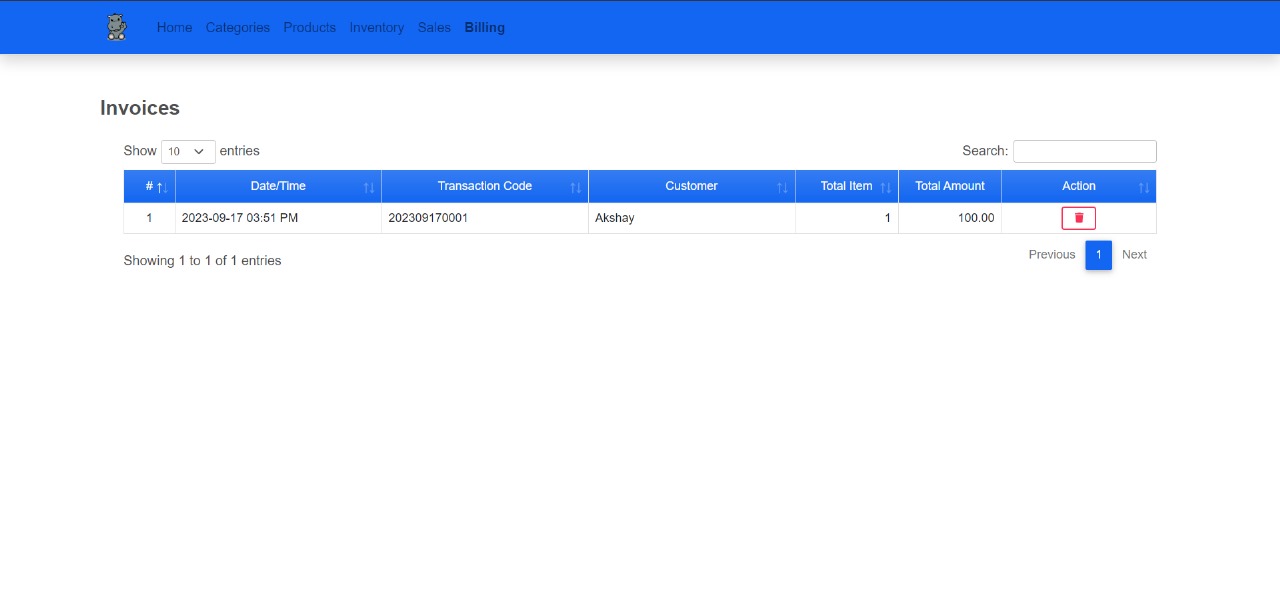
Product Inventory Details:



Billing Page:



Invoices Page:



**Conclusion**

In conclusion, the Django Pharmacy Management System represents a comprehensive solution tailored to meet the diverse needs of modern pharmaceutical operations. Through meticulous design and implementation, this system addresses key facets of pharmacy management, including category organization, product handling, inventory control, and seamless billing processes.

By leveraging Django's robust framework, we've crafted a flexible and scalable architecture that streamlines day-to-day operations within a pharmacy. The modular code structure, comprising well-defined models, views, templates, and forms, ensures clarity, maintainability, and extensibility. This architecture not only facilitates efficient management of inventory and sales but also allows for future enhancements and customizations as per evolving business requirements.

Moreover, the emphasis on user-friendly interfaces and intuitive design principles ensures accessibility for users of varying technical expertise. This user-centric approach aims to enhance usability, empowering pharmacy staff to navigate the system effortlessly, thereby boosting productivity and minimizing errors in daily tasks.

The system's core functionalities, from category management to inventory tracking and billing, stand as a testament to its capability in optimizing pharmacy operations. With its robust features, the Django Pharmacy Management System promises to revolutionize how pharmacies operate, offering a cohesive platform that fosters accuracy, efficiency, and improved customer service.

In essence, this project marks a significant step toward modernizing and enhancing the efficiency of pharmaceutical management. Its successful implementation not only aligns with current industry standards but also lays the groundwork for future innovations and advancements in pharmacy operations.

**Bibliography**

Certainly, creating a bibliography for a project like the Django Pharmacy Management System might include various sources such as online documentation, textbooks, articles, and possibly official documentation for the technologies used. Here's an example:

1. Django Documentation. [Online] Available at: https://docs.djangoproject.com/en/stable/

- The official documentation provided a foundational understanding and reference for implementing Django-based solutions.

2. Python Software Foundation. "Python Language Reference." Python.org.

- Understanding Python's syntax and functionalities was crucial for developing the Django-based system.

3. Flanagan, David, and Bill Frakes. "JavaScript: The Definitive Guide." O'Reilly Media, 2020.

- Insight into JavaScript, which might have been used for frontend interactions or dynamic web features within the system.

4. Martin, Robert C. "Clean Architecture: A Craftsman's Guide to Software Structure and Design." Prentice Hall, 2017.

- Concepts from this book might have influenced the design and structuring of the codebase to ensure maintainability and scalability.

5. Articles from academic journals or online platforms discussing pharmacy management systems, inventory control, and efficient billing processes.

- Academic or industry-specific articles could have provided insights into best practices and innovations in pharmacy management.