A MINI PROJECT REPORT ON

COURIER CONNECT USING DJANGO

Submitted in partial fulfilment for the award of the degree of

BACHELOR OF TECHNOLOGY

In

Computer Science and Engineering

By

K.N.S.L. Padma Janaki (21A81A0520)

B. Rohit Vinay Naga Chandra Reddy(21A81A0507)

P.V. V. Nagendra (21A81A0537)

Harshad Krishna. U (21A81A0518)

U. Raj Kumar(21A81A0557)

Under the Esteemed Supervision of

Mrs. B. Sri Ramya, M. Tech

Sr. Asst Professor

Department of Computer Science and Engineering (Accredited by N.B.A.)
SRI VASAVI ENGINEERING COLLEGE(Autonomous)
(Affiliated to JNTUK, Kakinada)
Pedatadepalli,Tadepalligudem-534101, A.P 2023-24

SRI VASAVI ENGINEERING COLLEGE (Autonomous)

Department Of Computer Science and Engineering

Pedatadepalli,Tadepalligudem

Certificate

This is to certify that the Project Report entitled "**COURIER CONNECT USING DJANGO**" submitted by **K.N.S.L PADMA JANAKI (21A81A0520), B.R.V.N. CHANDRA REDDY (21A81A0507), P.V.V. NAGENDRA (21A81A0537), HARSHAD KRISHNA.U (21A81A0518), U. RAJ KUMAR (21A81A0557)** for the award of the degree of Bachelor of Technology in the Department of Computer Science and Engineering during the academic year 2023-2024.

Name of Project Guide

Mrs. B. SriRamya, M. Tech

Sr. Asst Professor.

Head of the Department

Dr. D Jaya Kumari M. Tech, Ph.D

Professor &HOD.

External Examiner

DECLARATION

We hereby declare that the project report entitled "**Courier connect using Django"** submitted by us to Sri Vasavi Engineering College(Autonomous), Tadepalligudem, affiliated to JNTUK Kakinada in partial fulfilment of the requirement for the award of the degree of B. Tech in Computer Science and Engineering is a record of Bonafide project work carried out by us under the guidance of **Mrs. B. Sri Ramya, M. Tech Sr. Asst Professor** We further declare that the work reported in this project has not been submitted and will not be submitted, either in part or in full, for the award of any other degree in this institute or any other institute or University.

Project Associates

K.N.S.L. Padma Janaki(21A81A0520)

B.R.V.N. Chandra Reddy(21A81A0507)

P.V.V. Nagendra(21A81A0537)

Harshad Krishna. U(21A81A0518)

U. Raj Kumar(21A81A0557)

ACKNOWLEDGEMENT

First and foremost, we sincerely salute to our esteemed institute **SRI VASAVI ENGINEERING COLLEGE,** for giving us this golden opportunity to full fill our warm dream to become an engineer.

Our sincere gratitude to **Mrs. B. Sri Ramya, M. Tech Sr. Asst Professor** our project guide Department of Computer Science and Engineering**,** for her timely cooperation and valuable suggestions while carrying out this project.

We express our sincere thanks and heartful gratitude to **Dr . D. Jaya Kumari**, Professor & Head of the Department of Computer Science and Engineering, for permitting us to do our project.

We express our sincere thanks and heartful gratitude to **Dr . G.V.N.S.R. Ratnakara Rao**, Principal, for providing a favourable environment and supporting us during the development of this project.

Our special thanks to the management and all the teaching and non-teaching staff members, Department of Computer Science and Engineering, for their support and cooperation in various ways during our project work. It is our pleasure to acknowledge the help of all those respected individuals. We would like to express our gratitude to our parents, friends who helped to complete this project.

Project Associates

K.N.S.L. Padma Janaki(21A81A0520)

B.R.V.N. Chandra Reddy(21A81A0507)

P.V.V. Nagendra(21A81A0537)

Harshad Krishna. U(21A81A0518)

U. Raj Kumar(21A81A0557)

ABSTRACT

Micro Entrepreneurs often face challenges in efficiently delivering parcels due to manual processes of managing customer details, order tracking, and delivery personnel. The Courier Management System (CMS) was developed as a solution to these issues, providing a centralized software system that streamlines the entire delivery process from order booking to final delivery. The Courier Management System is designed to be user - friendly, efficient , reliable, and cost-effective, providing a high-quality courier service to small businesses and private shops. It simplifies the management of customer details, order tracking ,courier personnel management, invoice , and payment management, while also offering valuable insights through its reporting module. The CMS is scalable and customizable ,enabling small businesses and private shops to tailor the system to their specific requirements and improve their delivery process, ultimately offering a better customer experience.

TABLE OF CONTENTS

CHAPTER - 1

INTRODUCTION

1.1 INTRODUCTION

Courier management system is a software application designed to help businesses manage their courier and delivery operations efficiently. It provides a comprehensive platform for managing the entire delivery process, from order placement to final delivery, including tracking and monitoring of the shipment's progress at every stage of the process. The system is designed to streamline the entire delivery process, providing realtime updates, and reducing the amount of time spent on manual data entry and communication between various parties involved in the delivery process. Additionally, courier management systems can help businesses increase their customer satisfaction by providing better transparency and visibility into the delivery process and allowing customers to track their shipments in real-time. The system also helps businesses to optimize their delivery routes, reduce delivery costs, and improve their overall operational efficiency.

1.2 Motivation

In today's fast-paced world, the need for efficient, reliable, and secure courier services has never been more critical. The courier industry plays a pivotal role in facilitating the movement of parcels, documents, and goods across the globe. However, there are several challenges and opportunities that have inspired the development of our courier management project .With the advent of e-commerce and the growing demand for online shopping, customers expect a seamless and hassle-free parcel delivery experience .Our project aims to address these evolving customer needs by providing a user-friendly platform for registering parcels, tracking shipments, and reporting issues .The courier industry relies on the efficient coordination of parcel pickups, deliveries, and tracking .Our system offers delivery personnel and managers the tools they need to optimize operations and streamline the delivery process .Our courier management project aspires to transform the way people send and receive parcels, enhancing convenience, efficiency, and security in the courier industry. It is our belief that the future of courier services lies in user-centric, technology-driven solutions, and we are excited to be at the forefront of this transformative journey.

1.3 Scope:

Order management: managing orders from customers and assigning them to drivers.

Real-time tracking: tracking the location and status of packages and drivers.

Delivery scheduling: scheduling the delivery routes and time slots for drivers.

Driver management: managing the drivers' profiles, performance.

Warehouse management: managing the inventory and storage of packages in warehouses.

Customer communication: communicating with customers via SMS, email, or phone about their orders and delivery status.

1.4 Project Outline:

The Courier Connect is designed to optimize and simplify courier services by providing a user-centric platform for customers, managers, and delivery personnel. This project aims to offer a seamless experience for customers, enabling them to easily register parcels, track deliveries, and interact with the system. Managers will have the tools needed to efficiently oversee parcel operations within their branch, and delivery boys will benefit from a streamlined process for handling deliveries. The project leverages a modern technological framework to ensure the reliability and efficiency of the courier management process, prioritizing user satisfaction and operational excellence.

CHAPTER - 2

LITERATURE SURVEY

LITERATURE SURVEY

Title of the paper: An Integrated Courier Services Application

year:2018

Authors: mohammad sulaiman, Wong Yi Leng

Description: The integrated courier service application implements the concept of service innovation to solve inconvenience of courier service users of getting quotation and the need to travel to post office to drop their items

Title of the paper: Automatic Courier Management System

Publication year: November 1995

Authors: Naveen Durai, Nivetha M, Santhosh V

Description: The System that is developed helps the customer to find the location of the parcel. It helps them to know the parcel departure status. This added feature makes the user to conveniently use the system

Title of the paper: Courier Management System using Cloud

Publication year: October 2020

Authors: CK Vignesh

Description: This Project has solved the problems caused due to centralization and inefficient updating of a traditional courier management system and has additionally used the help of cloud computing to enable scaling and load balancing to enable High Availability and Fault Tolerance

Title of the paper: Online Courier Management System

Publication year: April 2023

Authors: Rahul Devadiga, Deeshitha V, Pradeesh, Rashmi Gavadi

Description: The System is designed to be user-friendly, secure, and efficient, with a focus on meeting the requirements of users. In the future, the system can allow for further enhancement and application development to meet the evolving needs of courier management.

CHAPTER - 3

SYSTEM STUDY AND ANALYSIS

3.1 PROBLEM STATEMENT

The problem statement describes the challenges faced by businesses in managing their courier and delivery operations. These challenges include manual data entry, inefficient delivery operations, lack of real-time tracking, high delivery costs, poor communication, and security concerns. These issues can lead to errors, delays, additional costs, lost packages, dissatisfied customers, and lower profit margins. The goal of the project is to develop a solution that addresses these challenges and improves the delivery process for businesses.

3.2 EXISTING SYSTEM

The current courier system has some significant drawbacks. One of the major issues is the inconvenience it causes to the users. The requirement to physically visit the courier office can be time-consuming and may also involve transportation costs, particularly for users who live far away from the office. Overall, the current courier system is not user-friendly, and the lack of an online platform and real-time tracking system can cause inconvenience, uncertainty, and delays for the users.

3.3 LIMITATIONS OF EXISTING SYSTEM

- Inability to Manage Multiple Customers
- Delayed Deliveries
- Fully Dependency on Human Resources
- High Delivery Cost
- There is no optimal route for transmitting the courier

3.4 PROPOSED SYSTEM

- This courier system allows users to send their courier from the comfort of their own homes without the need to visit a physical courier office. The user's data will be stored in a secured database ensuring privacy and confidentiality.
- Users can track their courier in real-time and receive timely updates as it reaches each checkpoint.

- The admin has the authority to update the checkpoints and track the courier's movement.
- The system is designed to be user-friendly, efficient, and offers a hassle-free courier experience.

3.5 ADVANTAGES OF PROPOSED SYSTEM

- Easy to register parcel from any where
- Easy to track the status of applications at any level at any point in time
- The centralized database helps in avoiding conflicts
- Easy to use GUI that does not require specific training.

3.6 FUNCTIONAL REQUIREMENTS

Functional requirements, also known as system requirements, define the functions that a system must perform in order to meet the needs of its users. They describe what the system should do and how it should behave under specific conditions.

Customer

- Signup/Login
- Register parcel
- Track parcel
- Report issues

Delivery Boy

- Login
- Update parcel status
- Pick up and deliver parcels

Branch Manager

- Monitor and manage parcels
- Generate tracking IDs
- Mail tracking IDs to customers
- View reports

Admin

- Manage users

3.7 NON-FUNCTIONAL REQUIREMENTS

Non-functional requirements define the operational characteristics of a system, often representing the quality attributes or constraints. These requirements describe how a system should perform, rather than what it should do.

- Usability
- Performance
- Security
- Reliability
- Scalability
- Integration
- Compatibility

3.8 Software Requirements

- Operating System : Windows 8 and above
- Front-end: HTML,CSS,JAVASCRIPT
- Back-end: Django
- Database: SQLITE
- IDE : visual studio code

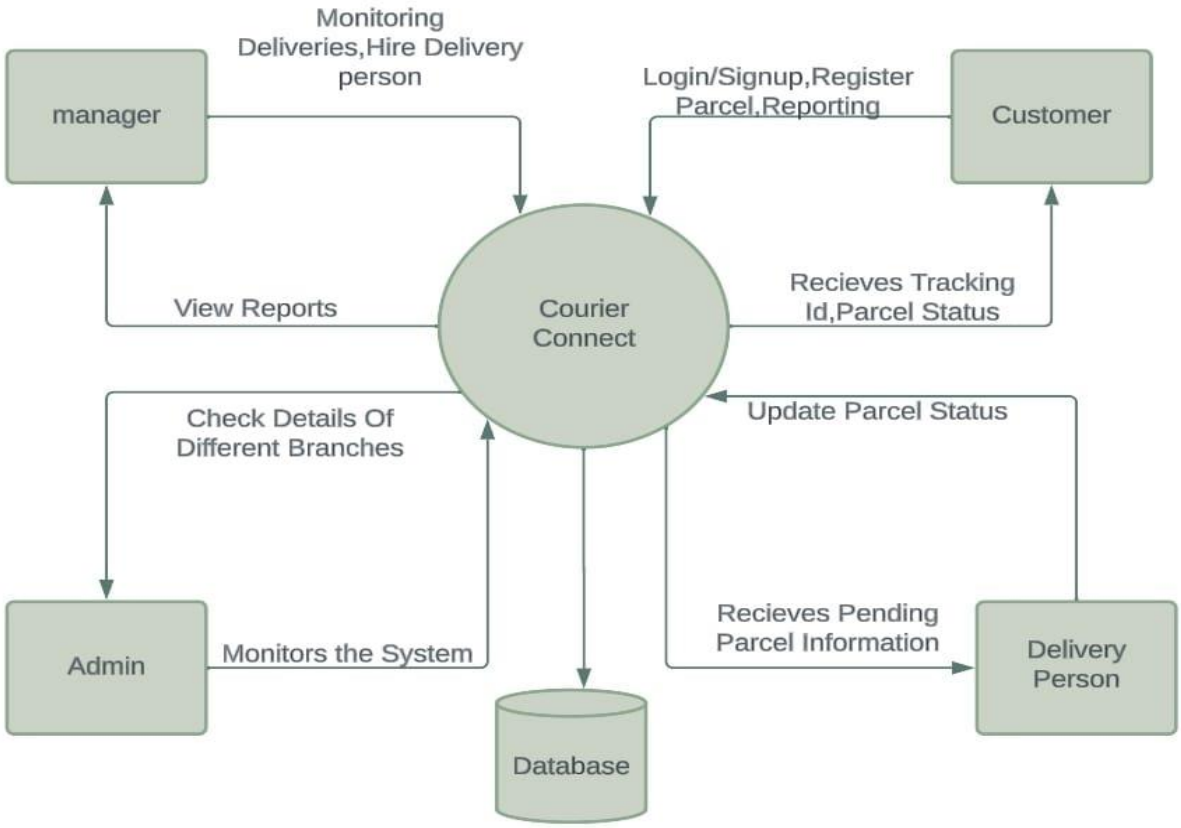
3.9 Hardware Requirements

- Intel Core processor i5
- RAM 8GB and Above
- SSD 256 GB and Above

CHAPTER - 4

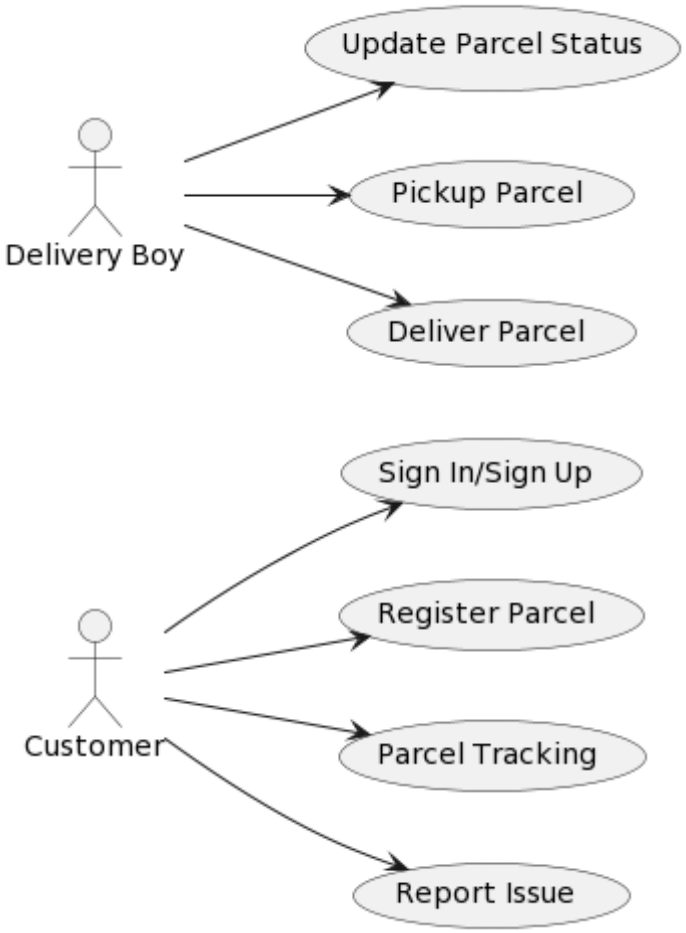
SYSTEM DESIGN

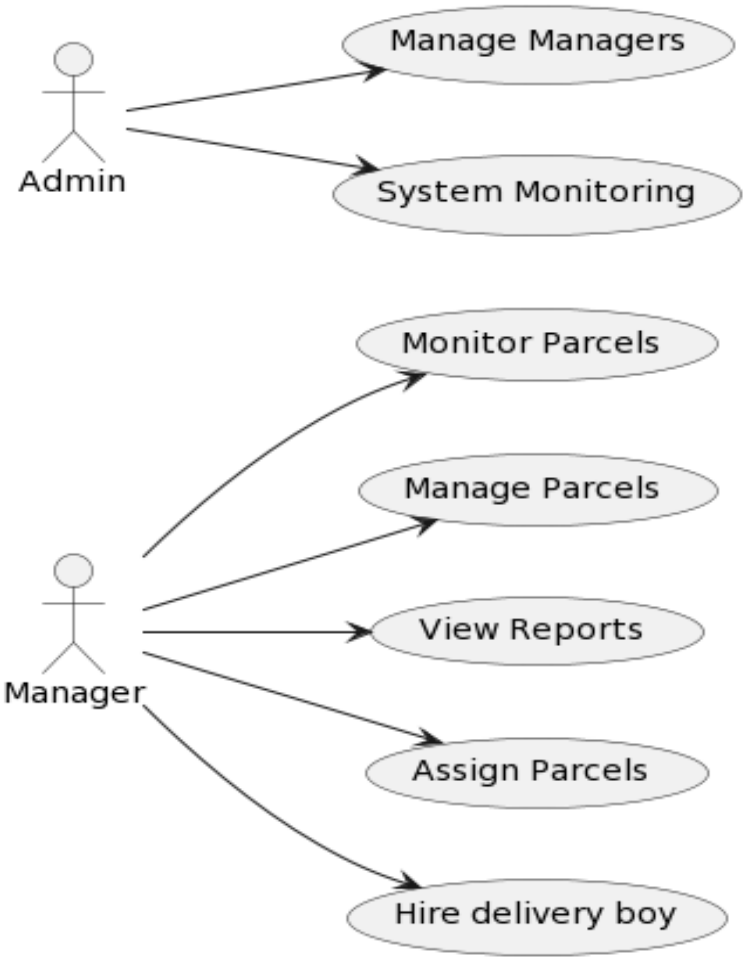
4.1 SYSTEM ARCHITECTURE



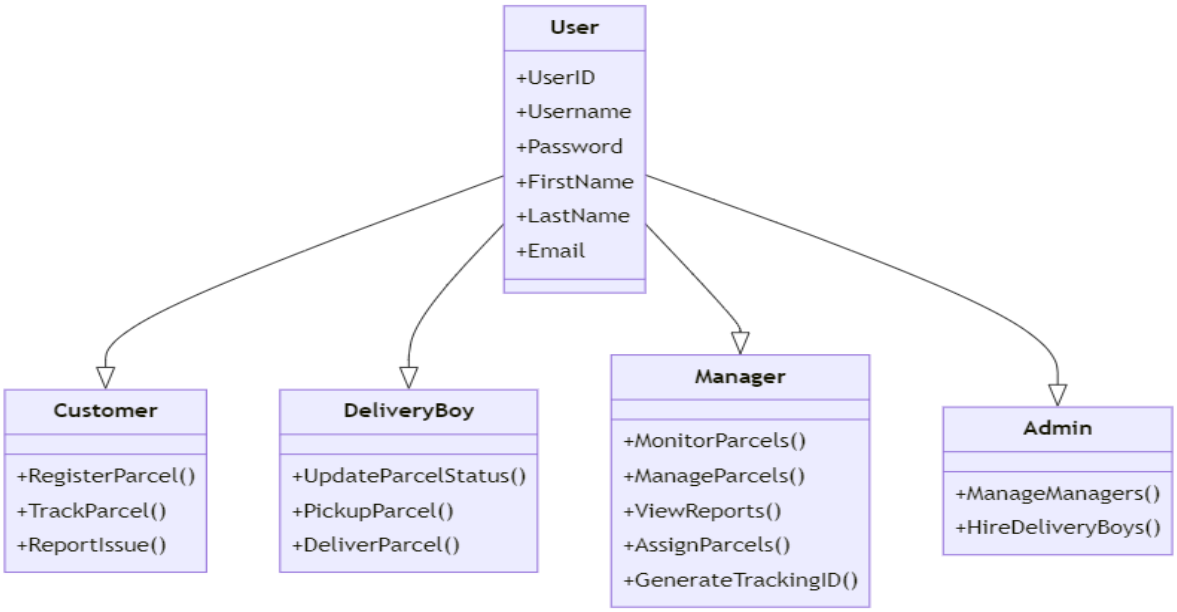
4.2 UML Diagrams:

4.2.1 UseCase Diagram

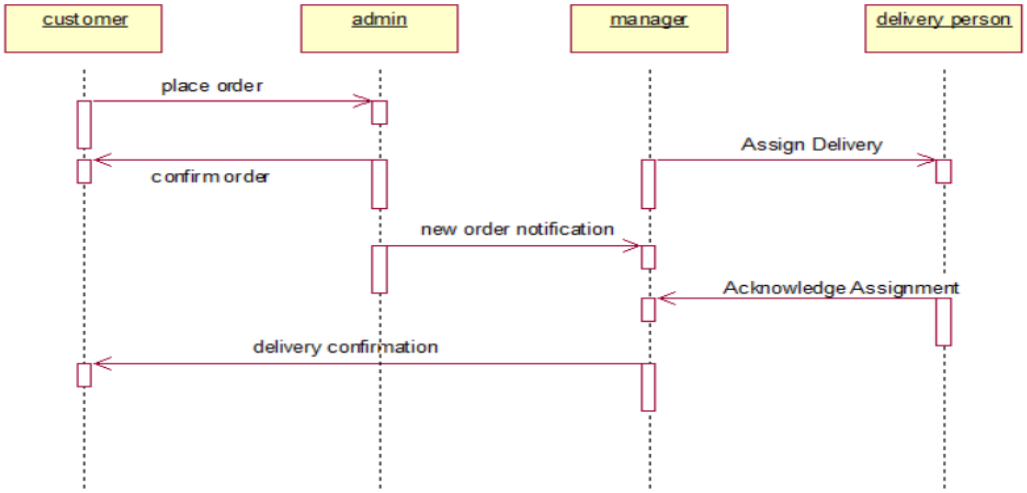




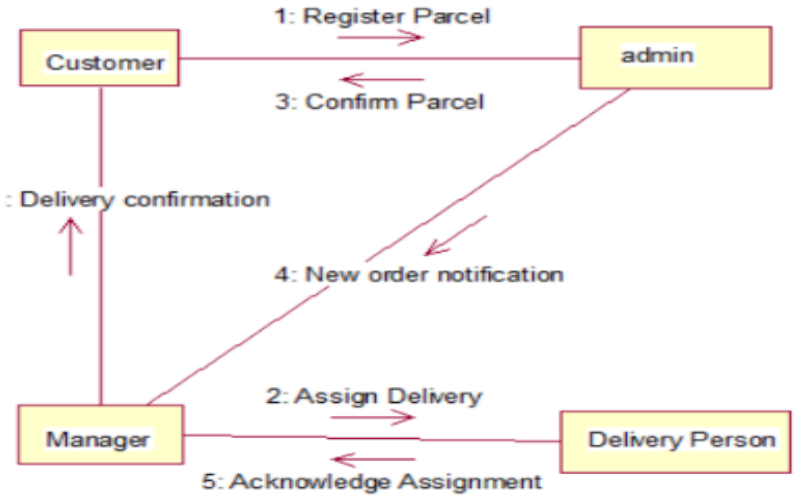
4.2.2 Class Diagram



4.2.3 Sequence diagram:



4.2.4 Collaboration Diagram:



CHAPTER - 5

TECHNOLOGIES

5.1 HTML

Introduction:

HTML (Hypertext Markup Language) uses a markup system composed of elements which represent specific content. Markup means that with HTML you declare what is presented to a viewer, not how it is presented. Visual representations are defined by Cascading Style Sheets (CSS) and realized by browsers. Still existing elements that allow for such, like e.g. font, "are entirely obsolete, and must not be used by authors".

HTML is sometimes called a programming language but it has no logic, so is a

markup language.HTML tags provide semantic meaning and machine-readability to

the content in the page.

An element usually consists of an opening tag (<element name>), a closing tag (</element name>), which contain the element's name surrounded by angle brackets, and the content in between:

<element name>...content...</element name>.

Features of HTML:

It is easy to learn and easy to use.

It is platform-independent.

Images, videos, and audio can be added to a web page.

Hypertext can be added to the text.

It is a markup language.

5.2 CSS

Introduction:

Cascading Style Sheets (CSS) is a style sheet language used for describing the presentation of a document written in a markup language such as HTML or XML.CSS is designed to enable the of presentation and content, including layout, colors, and fonts. This separation can improve content accessibility; provide more flexibility and control in the specification of presentation characteristics; enable multiple web pages to share formatting by specifying the relevant CSS in a separate css file, which reduces complexity and repetition in

the structural content and enable the .css file to be cached to improve the page load speed between the pages that share the file and its formatting.

The name cascading comes from the specified priority scheme to determine which style rule applies if more than one rule matches a particular element. This cascading priority scheme is predictable.

Features of CSS:

CSS can define color, font, text alignment, size, borders, spacing, layout and many other typographic characteristics, and can do so independently for on-screen and printed views. CSS also defines non-visual styles, such as reading speed and emphasis for aural text readers.

5.3 JavaScript

Introduction:

JavaScript often abbreviated JS, is a programming language that is one of the core technologies of the World Wide Web, alongside HTML and CSS. As of 2022, 98% of websites use JavaScript on the client side for web page behaviour, often incorporating third-party libraries. All major web browsers have a dedicated JavaScript engine to execute the code on user's devices. JavaScript engines were originally used only in web browsers, but are now core components of some servers and a variety of applications. The most popular runtime system for this usage is Node.js .Although Java and JavaScript are similar in name, syntax, and respective standard libraries, the two languages are distinct and differ greatly in design.

Features of JavaScript:

Validating User’s Input

Simple Client-side Calculations

Greater Control

Platform Independent

Handling Dates and Time

Generating HTML Content

Detecting the User’s Browser and OS

5.4 PYTHON

Python is a high level, interpreted and general-purpose dynamic programming language that focuses on code readability. It has fewer steps when compared to Java and C. It was founded in 1991 by developer Guido Van Rossum. It is used in many organizations as it supports multiple programming paradigms. It also performs automatic memory management.

Python is a high-level, interpreted, interactive and object-oriented scripting language. Python is designed to be highly readable. It uses English keywords frequently where as other languages use punctuation, and it has fewer syntactical constructions than other languages.

Python is a must for students and working professionals to become a great Software Engineer especially when they are working in Web Development Domain.

Advantages of Python:

1. **Readability:** Its clean and readable syntax makes it easy to understand and write, reducing the cost of program maintenance and development.
2. **Vast Libraries:** Python has an extensive standard library and a wide range of third-party libraries for various applications, facilitating rapid development.
3. **Diverse Usage:** It can be used for web development, scientific computing, artificial intelligence, automation, data analysis, and more.
4. **Community Support:** The Python community is vibrant and large, providing extensive documentation, tutorials, and forums for support and collaboration.
5. **Interpreted Language:** Python's interpreted nature allows for quick development and prototyping, making it an ideal language for many projects.
6. **Platform Independent:** It’s available on various platforms, making it highly portable.
7. **Scalability:** Python scales well and is used in both small-scale and large-scale applications.
8. **Productivity:** Due to its simplicity and ease of use, development in Python is typically faster than in other languages.
9. **Integration Capabilities:** It can be easily integrated with other languages and tools, making it a great glue language for heterogeneous environments.
10. **Open Source:** Python is open-source, encouraging community-driven development and innovation.

There are various frameworks of python like:

- Bottle
- Flask
- Django
- Web2py
- AIOHTTP
- CherryPy
- Dash
- Falcon
- Growler
- UvLoop
- Pyramid
- Sanic
- CubicWeb
- TurboGears
- Hug
- MorePath

We are using Django for our project among these frameworks.

5.5 Django:

Django is a high-level Python web framework that encourages rapid development and clean, pragmatic design. Built by experienced developers, it takes care of much of the hassle of web development, so you can focus on writing your app without needing to reinvent the wheel. It’s free and open source.

Django introduction

In this first Django article, we answer the question "What is Django?" and give you an overview of what makes this web framework special. We'll outline the main features, including some of the advanced functionality that we won't have time to cover in detail in this module. We'll also show you some of the main building blocks of a Django application (although at this point you won't yet have a development environment in which to test it).

Prerequisites: Basic computer literacy. A general understanding of Server-side website programming and in particular the mechanics of client server interactions in websites.

Objective: To gain familiarity with what Django is, what functionality it provides, and the main building blocks of a Django application

Django history:

Django was created in the fall of 2003, when the web programmers at the *Lawrence Journal-World* newspaper, Adrian Holovaty and Simon Willison, began using Python to build applications. Jacob Kaplan-Moss was hired early in Django's development shortly before Simon Willison's internship ended. It was released publicly under a BSD license in July 2005. The framework was named after guitarist Django Reinhardt. Adrian Holovaty is a Romani jazz guitar player and a big fan of Django Reinhardt. In June 2008, it was announced that a newly formed Django Software Foundation (DSF) would maintain Django in the future.

The Django admin site:

One of the most powerful parts of Django is the automatic admin interface. It reads metadata from your models to provide a quick, model-centric interface where trusted users can manage content on your site. The admin's recommended use is limited to an organization's internal management tool. It's not intended for building your entire front end around.

The admin has many hooks for customization, but beware of trying to use those hooks exclusively. If you need to provide a more process-centric interface that abstracts away the implementation details of database tables and fields, then it's probably time to write your own views.

In this document we discuss how to activate, use, and customize Django's admin interface.

Django officially supports the following databases:

- PostgreSQL, MySQL
- Oracle, SQLite

.

CHAPTER - 6

IMPLEMENTATION

6.1 MODULES:

6.1.1 Customer Module:

Parcel Management

Register a Parcel: Customers can create new delivery orders by providing details such as sender/receiver information, pickup and delivery addresses, and parcel details.

Track a Parcel: Customers can check the status and location of their parcels by providing a unique tracking number.

User Account Management

Sign Up: New customers can create accounts with their contact information.

Log In: Registered customers can log in to access their account.

6.1.2 Manager Module:

View All Parcels: Managers can access and view all parcels within their branch.

User Account Management

Log In: Managers can log in with their credentials.

6.1.3 Delivery Boy Module:

Parcel Assignment: Delivery boys can view and accept parcels assigned to them for delivery.

User Account Management

Log In: Delivery boys can log in with their credentials.

6.2 Code:

Models.py :

```
from django.db import models

import uuid
from django.contrib.auth.models import AbstractUser
from django.db.models.signals import post_save
from django.dispatch import receiver
from django.core.mail import send_mail


class User(AbstractUser):
    manager=models.BooleanField(default=False,null=True)
    worker=models.BooleanField(default=False,null=True)
    email=models.EmailField(unique=True,null=True)
    REQUIRED_FIELDS = []

    def __str__(self):
        return self.username
class Courier(models.Model):
        name=models.CharField(max_length=20)
        customer= models.ForeignKey(User,on_delete=models.CASCADE)
        destination=models.TextField()
        id=models.UUIDField(primary_key=True,default=uuid.uuid4,unique=True,editable=False)
        courier_id=models.CharField(max_length=50,default=uuid.uuid4)
        created=models.DateTimeField(auto_now=True)
        status=models.CharField(max_length=10,default="In Transit")
        delivery_by=models.CharField(max_length=20,default=None,null=True,blank=True)
        def __str__(self):
            return self.name[:20]


class ContactMessage(models.Model):
    name1 = models.CharField(max_length=100)
    email = models.EmailField()
```

```
    message = models.TextField()

    def __str__(self):
        return self.name1
class Courierform(models.Model):
    namee = models.CharField(max_length=255, blank=True, null=True)
    contactPhone = models.CharField(max_length=20)
    pickupAddress = models.TextField()
    sender_email = models.EmailField()
    receiver_email = models.EmailField()
    emergencyContactName = models.CharField(max_length=255, blank=True, null=True)
    emergencyContactPhone = models.CharField(max_length=20, blank=True, null=True)
    deliveryAddress = models.TextField(blank=True, null=True)
    pickupTime = models.CharField(max_length=100, blank=True, null=True)
    ampm = models.CharField(
    max_length=2,
    choices=[('am', 'AM'), ('pm', 'PM')],
    default='am')
    packageSize = models.CharField(
        max_length=10,
        choices=[('small', 'Small'), ('medium', 'Medium'), ('large', 'Large')],
        default='small')
    packageCategory = models.CharField(max_length=255, blank=True, null=True)
    packageWeight = models.FloatField()
    serviceType = models.CharField(
        max_length=20,
        choices=[('standard', 'Standard'), ('express', 'Express')],
        default='standard')
    privacyPolicyAgreement = models.BooleanField()
    termsAgreement = models.BooleanField()

    def __str__(self):
        return self.namee

class Report(models.Model):
```

```
    report= models.TextField(null=True)
    created=models.DateTimeField(auto_now=True)
    courier=models.ForeignKey(Courier, on_delete=models.CASCADE,null=True)
    customer= models.ForeignKey(User, on_delete=models.CASCADE,null=True)
    id=models.UUIDField(primary_key=True,default=uuid.uuid4,unique=True,editable=False)
    report_id=models.CharField(max_length=50,default=uuid.uuid4)
    def __str__(self):
        return self.customer.username[:20]

class Branch(models.Model):
    name=models.CharField(max_length=20)
    manager=models.ForeignKey(User, on_delete=models.CASCADE)
    address=models.TextField()
    id=models.UUIDField(primary_key=True,default=uuid.uuid4,unique=True,editable=False)
    branch_id=models.CharField(max_length=50,default=uuid.uuid4)
    couriers=models.ManyToManyField(Courier,related_name="couriers",blank=True)
    delivery=models.ManyToManyField(User,related_name="deliverys",blank=True)
    reports=models.ManyToManyField(Report,related_name="reports",blank=True)

    def __str__(self):
        return self.name[:20]

class Tracker(models.Model):
    courier=models.ForeignKey(Courier, on_delete=models.CASCADE,null=True)
    customer= models.ForeignKey(User, on_delete=models.CASCADE,null=True)
    present= models.ForeignKey(Branch, on_delete=models.CASCADE,null=True)
    def __str__(self):
        return self.courier.name[:20]

@receiver(post_save,sender=Courier)
def create_tracker(sender,instance,created,**kwargs):
    if created:
        Tracker.objects.create(courier=instance,customer=instance.customer)


def send_email_on_courier_creation(instance, created):
```

```
    if created:
        subject = 'New Courier Created'
        message = f'A new courier with name: {instance.name}, ID: {instance.courier_id}, and destination: {instance.destination} has been created.'
        from_email = 'courierconnect11@gmail.com'
        recipient_list = [instance.customer.email]
        send_mail(subject, message, from_email, recipient_list)


@receiver(post_save, sender=Courier)
def send_email_on_courier_creation_handler(sender, instance, created, **kwargs):
    send_email_on_courier_creation(instance, created)
```

views.py:

```
from django.shortcuts import render,redirect
from django.contrib.auth.models import User
from .models import Courier,Branch,Tracker,User,Report
from django.db.models import Q
from django.contrib.auth import login,logout
from django.contrib.auth.decorators import login_required
from django.http import HttpResponse
from .models import Courierform
from .models import ContactMessage
def home(request):
    if request.method == 'POST':
        sam=ContactMessage()
        sam.name1=request.POST['first']
        sam.email=request.POST['second']
        sam.message=request.POST['third']
        sam.save()
        return redirect('home')
    else:
        sam = None
        return render(request, 'home.html',{'sam':sam})


@login_required(login_url='login')
def courier_registration(request):
    if request.method == 'POST':
```

```
            form = Courierform()
            form.namee=request.POST['namee']
            form.contactPhone=request.POST['contactPhone']
            form.pickupAddress=request.POST['pickupAddress']
            form.sender_email=request.POST['senderEmail']
            form.emergencyContactName=request.POST['emergencyContactName']
            form.emergencyContactPhone=request.POST['emergencyContactPhone']
            form.receiver_email=request.POST['receiverEmail']
            form.deliveryAddress=request.POST['deliveryAddress']
            form.pickupTime=request.POST['pickupTime']
            form.ampm=request.POST['ampm']
            form.packageSize=request.POST['packageSize']
            form.packageCategory=request.POST['packageCategory']
            form.packageWeight=request.POST['packageWeight']
            form.serviceType=request.POST['serviceType']
            form.privacyPolicyAgreement=bool(request.POST.get('privacyPolicyAgreement'))
            form.termsAgreement=bool(request.POST.get('termsAgreement'))
            form.save()
            return render(request,"sucessp.html")
    else:
        form = None
    return render(request, 'booking.html', {'form': form})

def login_user(request):
    if request.method=="POST":
        email=request.POST.get("email")
        password=request.POST.get("password")
        try:
            user=User.objects.get(email=email)
        except:
            return render(request,"login.html",{"error":"Check Email!"})
        if user.check_password(password):
            login(request,user)
            if user.manager is False and user.worker is False:
                return redirect('track')
```

```
            elif user.worker is True:
                return redirect('shipments')
            else:
                return redirect('dashboard')
    return render(request,"login.html")

def logout_user(request):
    logout(request)
    return redirect('login')
def signup(request):
    if request.method=="POST":
        username=request.POST.get("username")
        email=request.POST.get("email")
        password=request.POST.get("password")
        try:
            user=User.objects.create_user(username=username,email=email,password=password)
            login(request,user)
            return redirect('login')
        except:
            return render(request,'signup.html',{'error':'Change email or username'})

    return render(request,'signup.html')


@login_required(login_url='login')
def manager(request):
    if request.user.manager==True:
        branch=Branch.objects.get(manager=request.user)
        courier_all=branch.couriers.all().order_by('-created')
        pending=courier_all.filter(status='pending')
        courier_all=courier_all.filter(status='Delivered')
        if request.method=="POST":
            client_id=request.POST.get("client_id")
            try:
                courier=Courier.objects.get(courier_id=client_id)
            except:
```

```
            return render(request,'dashboard.html',{'couriers':courier_all,'error':"something went to wrong"})
        if courier is not None:
            Tracker.objects.filter(courier=courier).update(present=branch)
            if courier.status=="In Transit" or courier.status=="pending":
                courier.status="pending"
                courier.save()
                branch.couriers.add(courier)
            else:
                return render(request,'dashboard.html',{'couriers':courier_all,'error':"already delivered"})

    return render(request,'dashboard.html',{'couriers':courier_all,'branch':branch,'todays':pending})
else:
    return HttpResponse("Not allowed")

@login_required(login_url='login')
def deliveryboy(request):
    try:
        branch=Branch.objects.filter(delivery=request.user)[0]
        couriers=Branch.objects.get(name=branch)
        couriers=couriers.couriers.filter(status="pending")
    except:
        couriers={}
    if request.method=="POST":
        client_id=request.POST.get("client_id")
        try:
            courier=Courier.objects.get(courier_id=client_id)
        except:
            return render(request,'shipments.html',{'couriers':couriers,'error':"Courier does not exist!!"})
        if courier is not None:
            courier.status="Delivered"
            courier.delivery_by=request.user.username
            courier.save()
    return render(request,'shipments.html',{'couriers':couriers})


@login_required(login_url='login')
```

```
def customer(request):
    if request.method=="POST":
        id=request.POST.get("tracking-number")
        try:
            tracker=Tracker.objects.get(courier__courier_id=id)
        except:
            return render(request,'track.html',{'error':'Check id!'})

        if request.user.id == tracker.courier.customer.id:
            return render(request,'track.html',{'tracker':tracker})
        else:
            return render(request,'track.html',{'error':'Not Allowed!'})
    return render(request,'track.html')

@login_required(login_url='login')
def hire_deliveryboy(request):
    branch=Branch.objects.get(manager=request.user)
    deliveryboys=branch.delivery.all()
    if request.method=="POST":
        email=request.POST.get("deliveryboy_email")
        try:
            user=User.objects.get(email=email.lower())
            try:
                newboy=branch.delivery.get(id=user.id)
                branch.delivery.remove(newboy)
            except:
                branchs=Branch.objects.filter(delivery=user)
                user.worker=True
                user.save()
                if len(branchs) !=0:
                    branchs[0].delivery.remove(user)
                branch.delivery.add(user)
            return render(request,'hiring_delivaryboy.html',{'deliveryboys':deliveryboys,'branch':branch})
        except:
```

```
                                                                                return
render(request,'hiring_delivaryboy.html',{'deliveryboys':deliveryboys,'branch':branch,'error':"user-not
exist"})
    return render(request,'hiring_delivaryboy.html',{'deliveryboys':deliveryboys,'branch':branch})


@login_required(login_url='login')
def view_reports(request):
    branch=Branch.objects.get(manager=request.user)
    reports=branch.reports.all()
    return render(request,'report.html',{'reports':reports})
@login_required(login_url='login')
def create_reports(request, id):
    try:
        courier = Courier.objects.get(courier_id=id)
    except Courier.DoesNotExist:
        return render(request, 'create_report.html', {'error': 'Courier not found'})

    if request.method == "POST":
        report_text = request.POST.get('report-body')


        try:
            branch = Tracker.objects.get(courier=courier)
        except Tracker.DoesNotExist:
            return render(request, 'create_report.html', {'error': 'Tracker not found'})
        if branch.present:
            report = Report.objects.create(report=report_text, courier=courier, customer=request.user)
            branch.present.reports.add(report)
            return render(request, 'create_report.html', {'courier': courier, 'success': 'Report added successfully'})
        else:
            return render(request, 'create_report.html', {'courier': courier, 'error': 'Branch is None'})


    return render(request, 'create_report.html', {'courier': courier})
def login_userr(request):
    if request.method=="POST":
        email=request.POST.get("email")
        password=request.POST.get("password")
```

```
        try:
            user=User.objects.get(email=email)
        except:
            return render(request,"login.html",{"error":"Check Email!"})
        if user.check_password(pafrom django.shortcuts import render,redirect
```

urls.py:

```
from django.contrib import admin
from django.urls import path,include
from . import views
urlpatterns =[
    path('',views. home,name="home"),
    path('booking/',views. courier_registration,name="booking"),
    path('login/',views.login_user,name="login"),
    path('dashboard/',views.manager,name="dashboard"),
    path('logout/',views.logout_user,name="logout"),
    path('signup/',views.signup,name="signup"),
    path('track/',views.customer,name="track"),
    path('shipments/',views.deliveryboy,name="shipments"),
    path('view-reports/',views.view_reports,name="report_manager"),
    path('create-reports/<str:id>',views.create_reports,name="create_reports"),
    path('hire_deliveryboy',views.hire_deliveryboy,name="hire_deliveryboy"),
     path('loginn/',views.login_userr,name="loginn")
]
```

CHAPTER - 7
TESTING

7.1 TESTING

System testing involves user training system testing and successful running of the developed proposed system. The user tests the developed system and changes are made according to their needs. The testing phase involves the testing of developed system using various kinds of data.

An elaborate testing of data is prepared and the system is tested using the test data. While testing, errors are noted and the corrections are made. The corrections are also noted for the future use. The users are trained to operate the developed system.

Testing involves operation of a system or application under controlled conditions and evaluating the results. The controlled conditions should include both normal and abnormal conditions. Testing should intentionally attempt to make things go wrong to determine if things happen when they shouldn't or things don't happen when they should .it is oriented to 'detection'

System testing is the stage of implementation that is aimed at ensuring that the system works accurately before live operation commences. Testing is vital to the success of the system. System testing makes logical assumption that if all the parts of the system are correct, then the goal will be successfully achieved. A series of testing are done for the proposed system before the system is ready for the user.

7.2Testing Methodologies

There are two major types of testing. They are

1. White Box Testing

2. Black Box Testing

7.2.1 White Box Testing

White box testing (also known as clear box testing, glass box testing, and transparent box testing and structural testing) is a method of testing software that tests internal structures are working of an application, an opposed to its functionality. In white box testing an internal perspective of the system, as well as programming skills, are used to design test cases. The tests are choosing inputs to exercise paths through the code and determine outputs

While white box testing can be applied at the unit, integration and system level of the software testing process, it is usually done at the unit level. It can test paths within a unit, path between units during integration, and between sub systems during system level tests.

This method of test design can uncover many of our problems it might not detect unimplemented parts of the specification or missing requirements.

White box test design techniques include:

- Control flow testing
- Data flow testing
- Branch testing
- Path testing
- Statement coverage
- Decision coverage

7.2.2 Black Box Testing

Black box testing is a method of software testing that examines the functionality of an (see white box testing). This method of test can be applied to virtually every level of software testing: unit, integration, system and acceptance. It typically compares most if not all higher level testing but can also dominate unit testing as well. The following are the types of testing:

1. Unit Testing

2. Integration Testing

3. Validation Testing

4. Verification Testing

5. User Acceptance Testing

1. Unit Testing

Unit testing focuses verification efforts and the smallest unit of the software design, the module. This is also known as “module testing”. The modules are tested separately, this testing was carried out during programming stage itself, in this testing each module is found to be working satisfactory as regards to the expected output from the module.

2. Integration Testing

Data can be lost across an interface: one module can have adverse efforts on another. Integration testing is the systematic testing for construction of program structure, while at the same time conducting test to uncover errors associated with in the interface. A correction is difficult because the isolation of the cause is complicated by the cast expanse of the entire program. Thus, in the integration testing step , all the errors uncovered are corrected for the next testing step

3. Validation Testing

At the conclusion of integration testing, software is completely assembled as a package, interfacing errors have been uncovered and corrected and a final series of software tests begins validation test has been conducted of the two possible conditions exists. One is the function or performance characteristics confirmed to specifications and are accepted and the other is deviation from specifications in uncovered and eficiency list is created.

4. Verification Testing

Verification is a fundamental concept in software design. This is the bridge between customer requirements and an implementation that specifies those requirements. This is verifiable if it can be demonstrated that the testing will result in an implementation that satisfies the customer requirements.

5. User Acceptance Testing

User acceptance testing of a system is the key factor of the success of the nay system. The system under study is tested for the user acceptance by constantly keeping in touch with theresuspective system users at any time of developing and making changes whenever required.

7.3 Test Cases:

7.3.1 Login:

Test Case ID	Description	Test Data	Expected Output
TC_01	Verify successful login with correct credentials	Valid email and password provided	Redirect to the Dashboard/Home page upon successful login.
TC_02	Verify login attempt with incorrect email format	Invalid email formats (e.g., missing @ or .com)	Display an error requesting a valid email format.
TC_03	Verify login attempt with incorrect password	Incorrect or invalid password provided	Display an error for incorrect password.
TC_04	Verify login attempt with blank fields	Leaving email or password field blank	Display an error requesting all required fields to be filled.
TC_05	Verify redirection to the signup page upon clicking 'Sign up'	Clicking the 'Sign up' link	Redirect to the Signup page.

7.3.2 Parcel Registration:

Test Case ID	Description	Test Data	Expected Output
TC_01	Verify all mandatory fields are filled	All fields except optional ones are filled.	Submit successful, data is sent to the server for processing.
TC_02	Verify invalid email format in the sender's email field	Invalid email formats (e.g., missing @ or .com)	Display an error prompting the user to enter a valid email.
TC_03	Verify valid phone number format for contact fields	Valid phone numbers with specific patterns	Accept the valid format, display an error for an invalid one.
TC_04	Verify invalid entries in size and weight fields	Non-numeric or excessive weight and size inputs	Display an error, prompt the user to enter valid numeric data.
TC_05	Verify submitting the form without agreeing to terms	All fields filled without checking the agreement checkboxes	Display an error, request to agree to the terms and policies.
TC_06	Verify that the reset button clears all form fields	All fields filled, reset button clicked	All fields should be cleared, returning the form to its initial state.

7.3.3 Tracking :

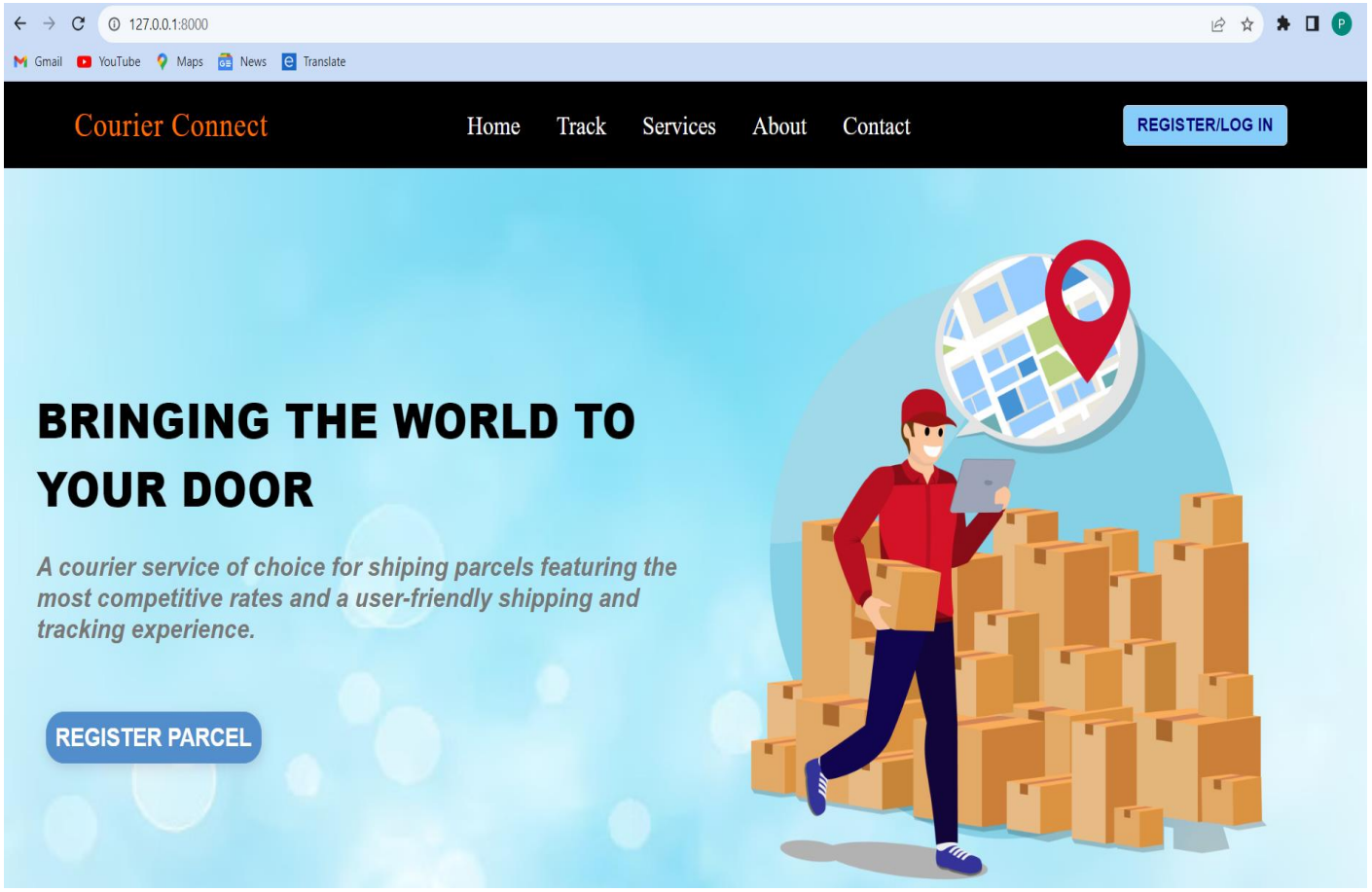
Test Case ID	Description	Test Data	Expected Output
TC_01	Verify tracking with a valid tracking number	Valid tracking number	Display the details of the parcel and its current status.
TC_02	Verify tracking with an invalid or non-existent number	Invalid tracking number or non-existent tracking ID	Display an error message indicating no details found.
TC_03	Verify tracking with an empty field	Leaving the tracking number field empty	Display an error requesting a valid tracking number.
TC_04	Verify creation of a report	Successful report creation for a valid tracking number	Display a success message for report creation.

.

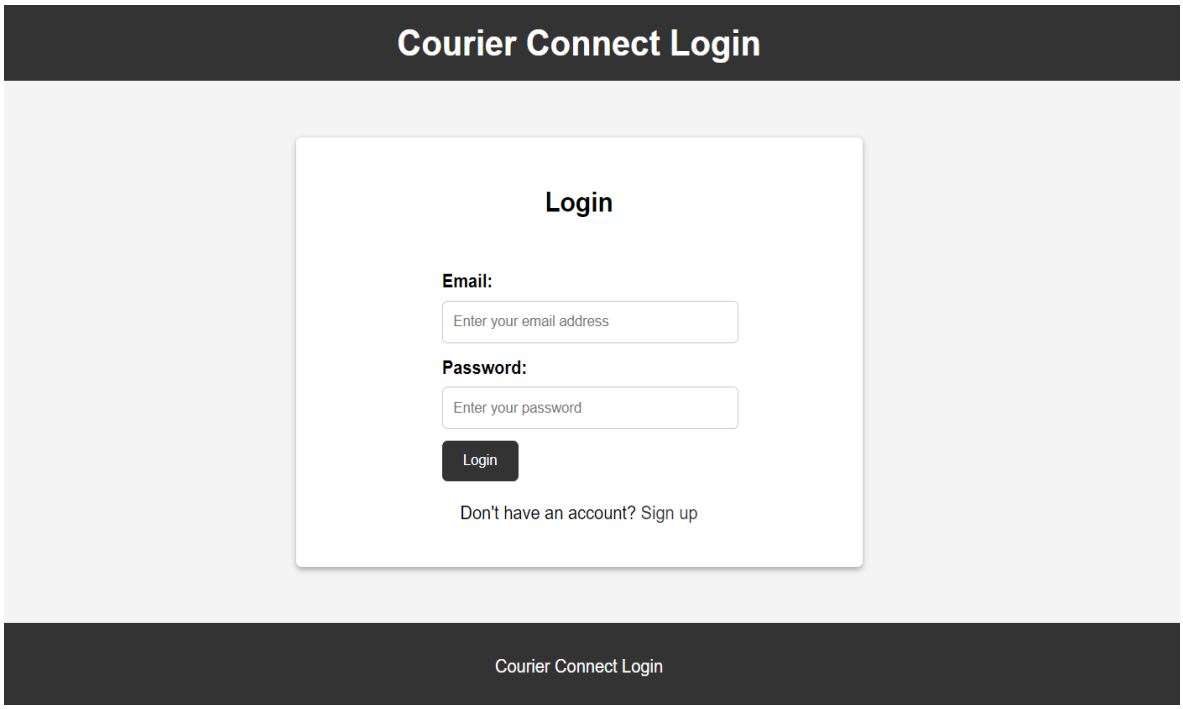
CHAPTER -8

SCREENSHOTS

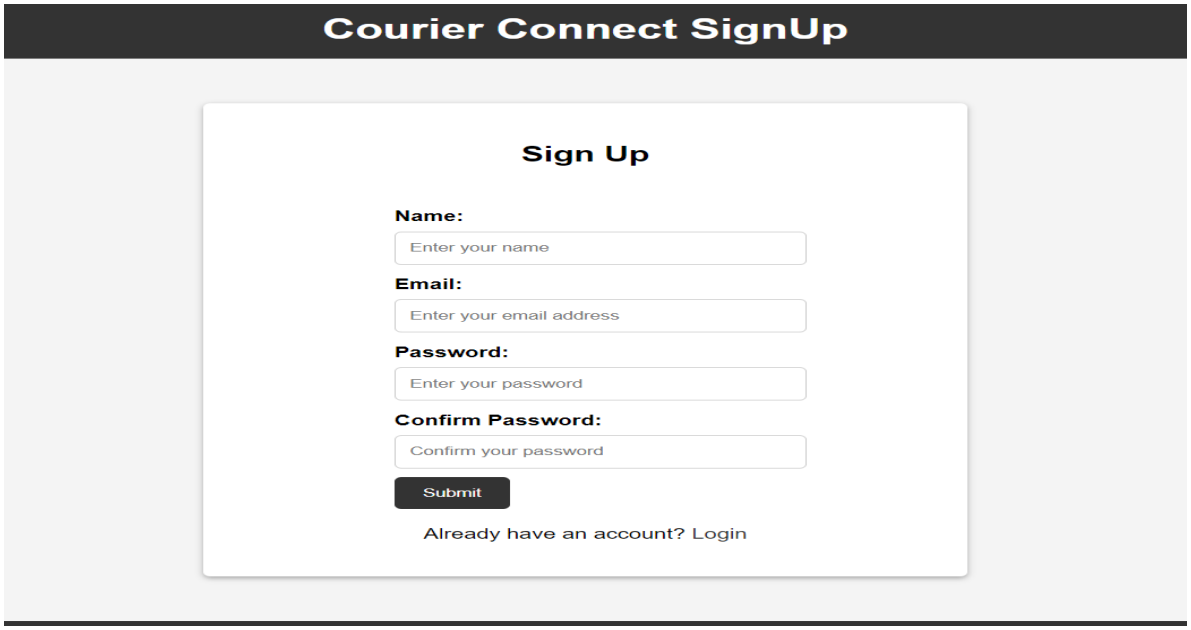
Home page:



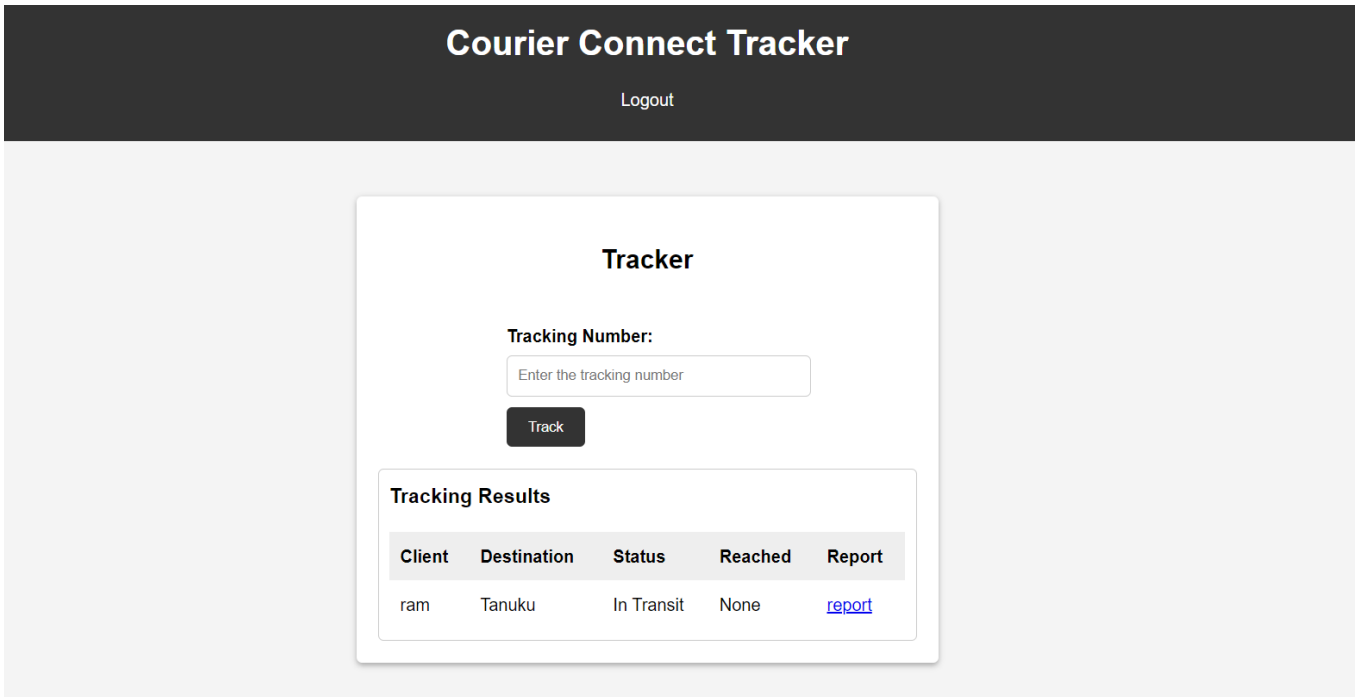
Login page:



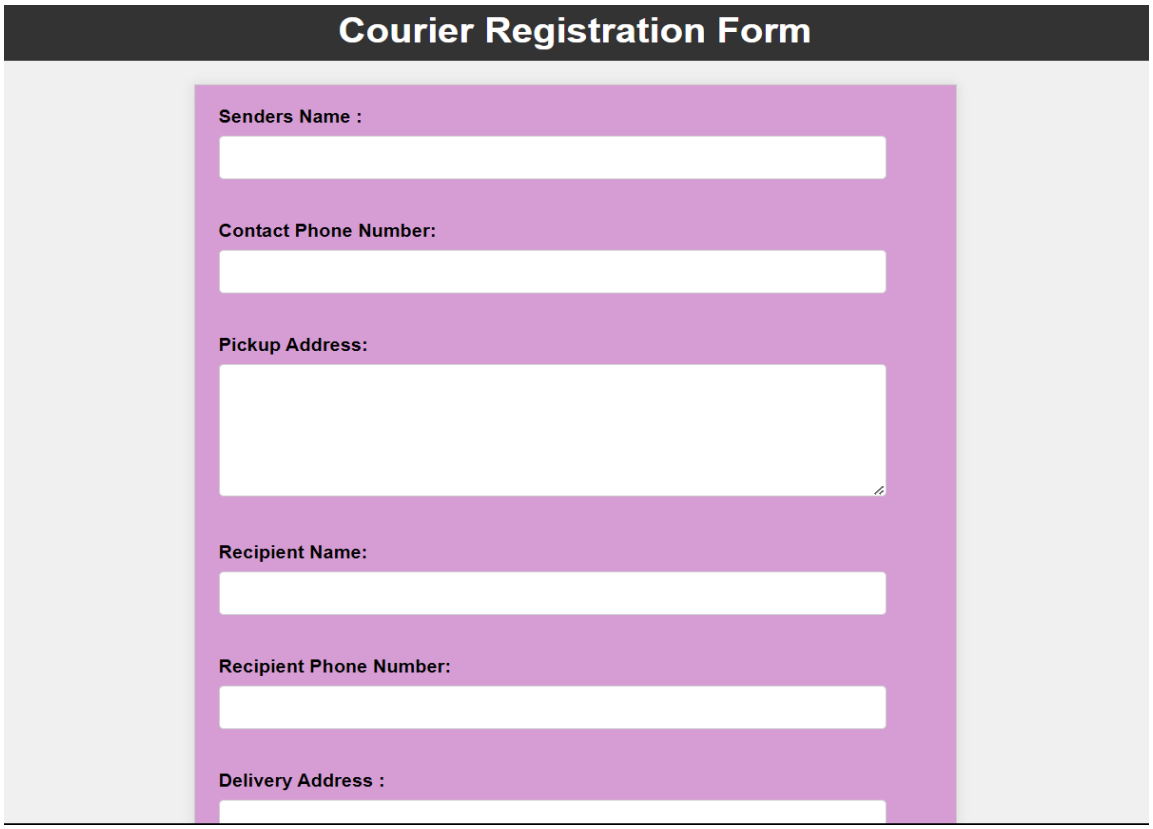
Sign up page:



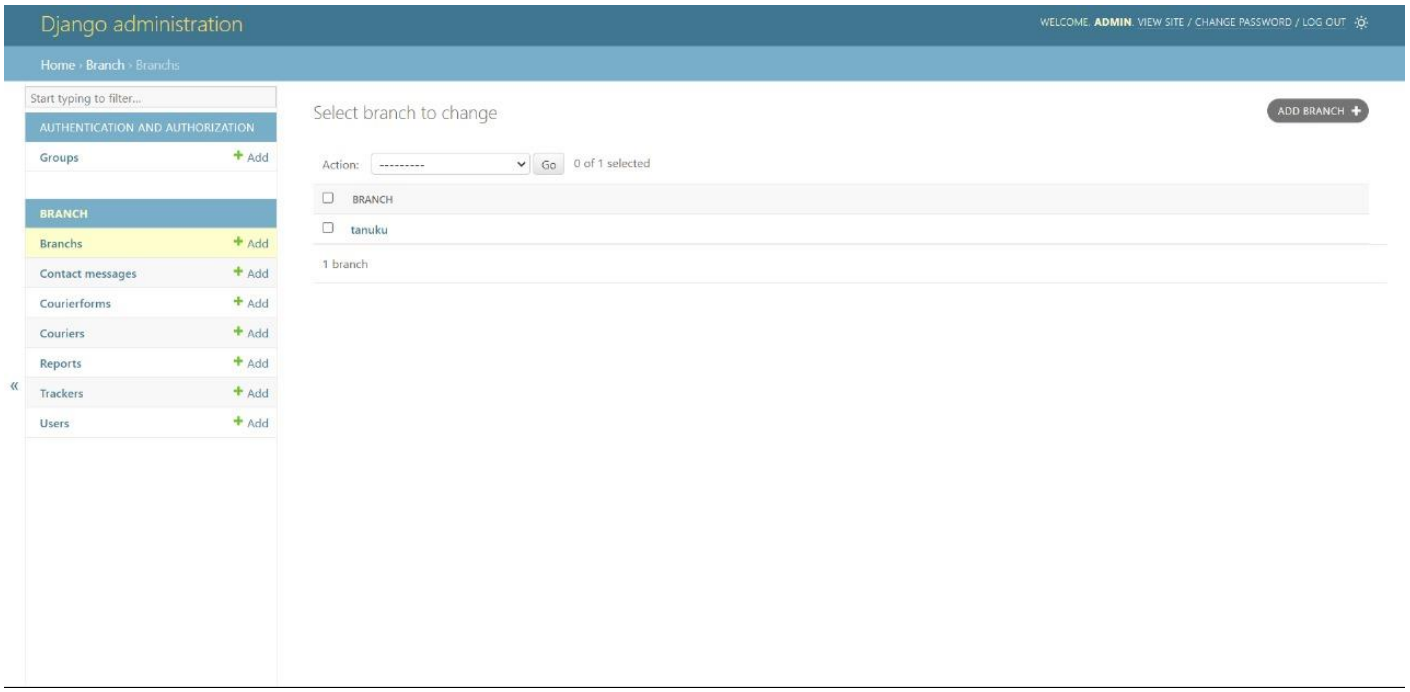
Tracking page:



Registration Form:

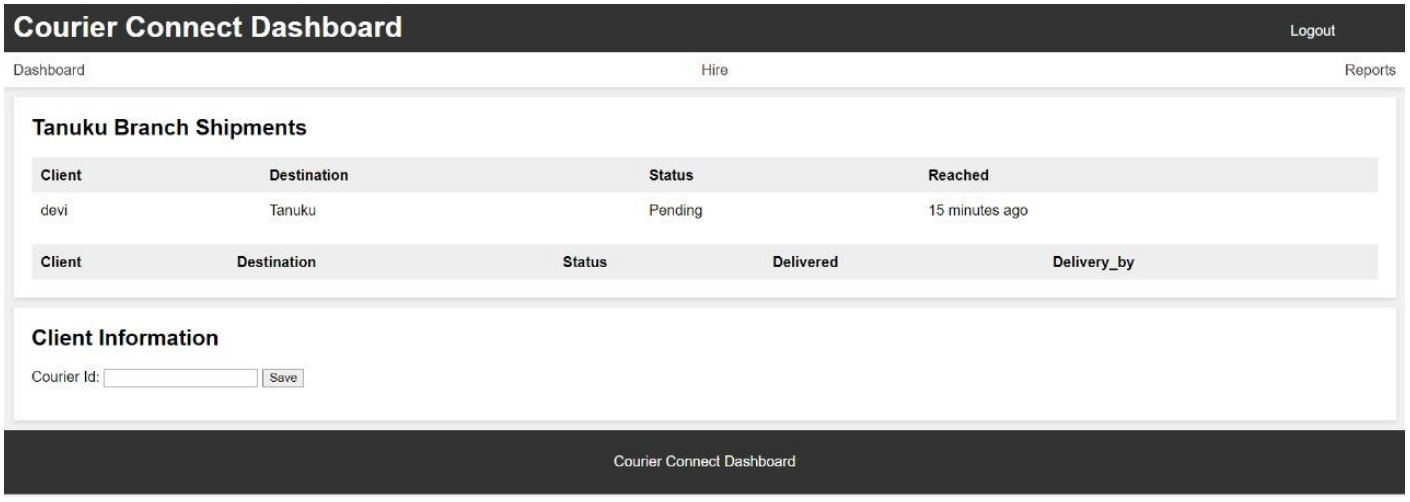


Admin page:

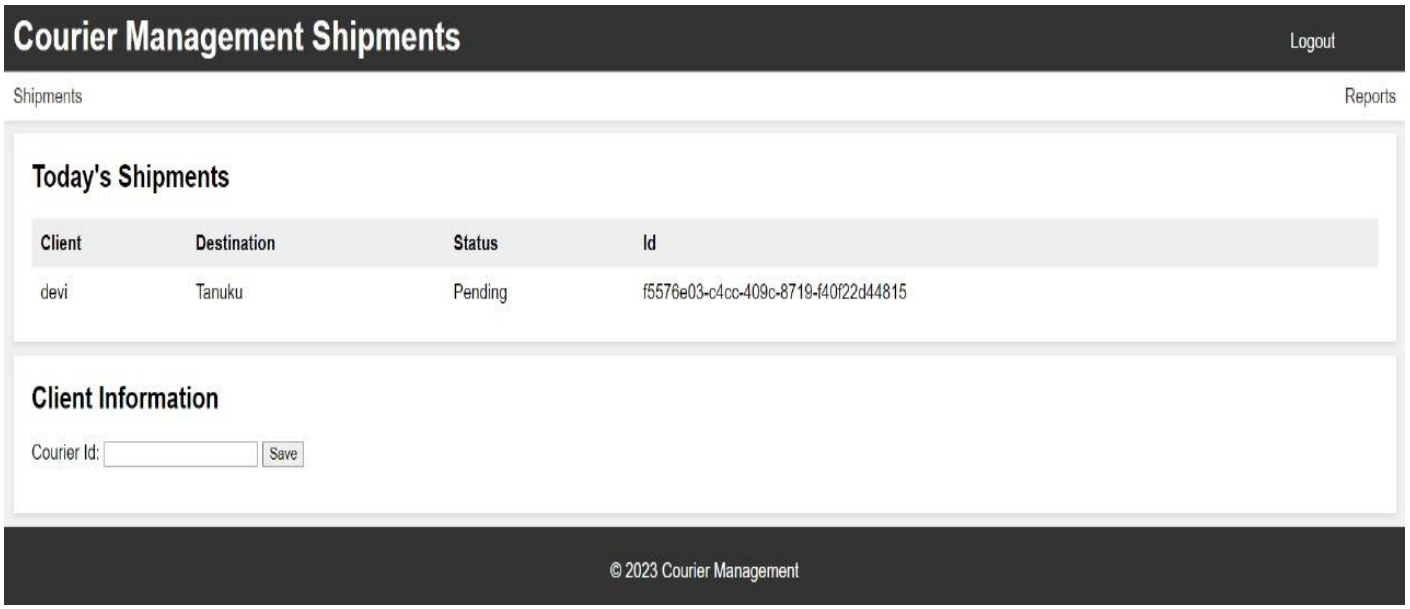


Courier tracking id to customer email :

Manager Dash Board:



Shipments Page:



CHAPTER - 9

CONCLUSION AND FUTURE WORK

9.1 CONCLUSION:

courier management system is a valuable tool for businesses that rely on courier services to deliver their products. By automating manual tasks, providing real-time tracking and reporting, and optimizing delivery routes, a courier management system can help businesses improve efficiency, increase productivity, enhance visibility, provide better customer service, and save on delivery costs. The specific outcomes of implementing a courier management system will depend on the unique needs of each business, the features of the software, and the implementation process. However, overall, a courier management system can be a significant asset to businesses looking to streamline their courier delivery process and improve their bottom line.

9.2 FUTURE WORK:

- **Route Optimization:** Incorporate route optimization algorithms to help delivery boys plan and execute their deliveries more efficiently, saving time and resources.
- **Customer Feedback System:** Integrate a feedback system that allows customers to rate delivery services, which can help in improving service quality.
- **Automated Notifications:** Enhance the notification system to provide real-time updates to customers via SMS or push notifications.
- **Machine Learning for Fraud Detection:** Implement machine learning algorithms to detect and prevent fraudulent activities, enhancing security.
- **Voice Assistants and AI Chatbots:** Allow users to interact with the system using voice commands and implement AI chatbots for customer support.

CHAPTER - 10

REFERENCES

REFERENCES:

[1] Rahul Devadiga, Deekshitha V, Pradeesh, Rashmi Gavadi. " Online Courier Management ". International Research Journal of Engineering and Technology (IRJET) e-ISSN: 2395-0056 Volume: 10 ,Issue: 04 , Apr 2023 www.irjet.net p-ISSN: 2395-0072

[2] Prof. Kshirsagar Sopan B , Kale Prasad S , Auti Saurabh S , Gopale Pratik B , Gopale Tejas S ."Courier Management System for e-commerce." International Journal of Research Publication and Reviews, Vol 3, no 5, pp 3454-3457, May 2022

[3] CK Vignesh." COURIER MANAGEMENT SYSTEM USING CLOUD COMPUTING" EPRA International Journal of Research and Development (IJRD) , ISSN: 2455-7838, Volume: 5 , Issue: 10 , October 2020

[4] S. Ammulu, K.Madhu Sudhan Reddy ." Online Courier Management System". IJIRT ,Volume 4 Issue 11 ,ISSN: 2349-6002, April 2018.

[5] Okemiri Henry A, Nweso Emmanuel Nwogbaga, Francis N. Nwebonyi" Critical Review Of Courier Service Management System With Empasis To Its Relevance If Adopted In Nigeria". Journal of Multidisciplinary Engineering Science and Technology (JMEST) ISSN: 2458-9403 Vol. 4 Issue 8, August - 2017.

[6] Marcel Kunkel and Michael Schwind." Cost and Market-based Pricing in the Courier Express and Parcel Service Industry " . IEEE Conference on Commerce and Enterprise Computing,2011