Project Report
on
Courier Management System

Submitted By

Ms. Piyusha Bhadange - 122B1F006
Ms. Revati Keskar - 123B2F149
Ms. Vaibhavi Kolhe - 123B2F150

Under the guidance of

Mrs. Sphurti Deshmukh

Department of Information Technology

Pimpri Chinchwad Education Trust's

Pimpri Chinchwad College of Engineering

Academic Year 2024-25

Abstract
The Courier Management System (CMS) is a software solution designed to optimize and manage logistics operations for a courier service provider. It handles three primary services: organization-to-organization delivery, personal package shipping, and warehousing. The system streamlines the process of transporting goods and documents between businesses, manages individual parcel deliveries with tracking and notifications, and integrates warehousing functions such as inventory management and dispatch. The system uses a centralized database to ensure efficient handling of transactions, real-time tracking, and reporting, enhancing operational efficiency and customer satisfaction. By automating and centralizing the key aspects of courier and warehousing services, the CMS reduces errors, increases reliability, and provides a scalable platform for growth.

Introduction

The Courier Management System (CMS) is a sophisticated software solution tailored to manage and streamline the entire logistics and delivery process for courier service providers. With the growing demand for faster, more efficient, and reliable delivery services in today's digital economy, businesses and individuals need robust systems that can handle various types of shipments, provide real-time tracking, and ensure smooth operations across all stages of the delivery lifecycle. The CMS addresses these needs by offering an integrated platform that supports a wide range of courier and warehousing services.

This system is designed to manage three primary service types: **Organization-to-Organization (Org-to-Org) Delivery, Personal Package Delivery**, and **Warehousing**.

- **Org-to-Org Delivery**: This service caters to business clients that require secure and efficient transport of goods, documents, and materials between organizations. It includes bulk delivery scheduling, invoicing, and real-time tracking features, ensuring a seamless and professional experience for businesses involved in inter-organizational shipments.

- **Personal Package Delivery**: The CMS also handles personal package delivery services, which cater to individual users looking to send or receive parcels. This service provides features such as express delivery options, real-time package tracking, notifications, and customer support, offering a convenient solution for individuals needing reliable courier services.

- **Warehousing**: In addition to delivery, the system integrates warehousing capabilities. It allows businesses to store, manage, and track inventory efficiently. With features such as automated stock-level monitoring, order dispatch scheduling, and warehouse space optimization, the CMS ensures that goods are handled and dispatched with minimal delays, improving overall supply chain efficiency.

The **Courier Management System** leverages a centralized database to handle large volumes of transactions, track shipments in real-time, and generate detailed reports for both customers and administrators. This database architecture ensures scalability, security, and efficient data processing, which is crucial for managing the fast-paced nature of logistics operations. By automating key processes, reducing human errors, and offering easy-to-use interfaces for both customers and service providers, the CMS enhances the overall user experience, minimizes operational inefficiencies, and helps businesses meet the increasing demands of modern courier and logistics services.

The system is designed to be highly adaptable, allowing for future expansion, such as integrating additional delivery options, incorporating advanced tracking technologies (e.g., GPS), or expanding warehouse management functionalities to handle higher volumes and more complex operations. By consolidating multiple services into one platform, the CMS simplifies the courier management process, reduces overhead costs, and ensures that customers and businesses can rely on timely, efficient, and cost-effective delivery solutions.

Scope:

The scope of the **Courier Management System** encompasses three core service areas: organization-to-organization (Org-to-Org) deliveries, personal package deliveries, and warehousing services. The system is designed to manage the entire lifecycle of courier services, including package pickup, tracking, delivery scheduling, invoicing, and warehousing. It supports real-time tracking of packages, providing visibility to both customers and couriers throughout the delivery process. Additionally, the CMS incorporates warehouse management functionalities, such as inventory tracking, order dispatching, and stock level monitoring, to improve storage and dispatch efficiency.

The system is scalable and can be extended to accommodate future growth, such as adding more delivery options, integrating with third-party logistics, or expanding warehouse capabilities.

Purpose:

The purpose of the **Courier Management System** is to provide an automated and user-friendly platform that facilitates the smooth operation of courier and logistics services. By centralizing data and operations, the system aims to reduce manual intervention, minimize errors, and improve overall service efficiency. It is designed to help businesses and individuals track and manage their deliveries in real-time, improving transparency and customer satisfaction. The CMS also aims to optimize warehousing operations, helping businesses efficiently store and distribute goods while minimizing delays and maintaining proper inventory control. Ultimately, the system is intended to support businesses in meeting the growing demands of fast, reliable, and cost-effective courier and logistics services.

Functional Requirements

1. User Management

- **Account Creation & Login**: The system should allow users (both business and personal customers) to create accounts, login, and manage their profiles securely.
- **Role-Based Access Control**: The system should support multiple user roles (e.g., admin, courier, customer, warehouse staff) with different access permissions.
- **User Dashboard**: Each user should have a personalized dashboard to view their orders, delivery statuses, and notifications.

2.Order Management

- **Create New Orders**: Users must be able to place new orders for both personal and business deliveries, specifying the type of service (Org-to-Org, personal, or warehousing).
- **Order Scheduling**: Customers should be able to schedule a pickup for the packages, with options for selecting delivery speed (e.g., standard, express).
- **Order Modification**: Customers can modify or cancel their orders before they are dispatched.

3.Delivery Management

- **Courier Assignment**: Admin users should be able to assign couriers to specific delivery routes and monitor their progress.
- **Proof of Delivery**: The system should support electronic proof of delivery (e.g., signature capture) to confirm successful deliveries.
- **Delivery Notifications**: Both customers and couriers should receive automated notifications regarding order status, such as pickup confirmation, in-transit updates, and successful delivery alerts.

4.Warehousing Management

- **Inventory Management**: The system should track and manage warehouse inventory, including adding new stock, tracking stock levels, and managing storage locations.
- **Warehouse Dispatch**: The system should support scheduling the dispatch of goods from the warehouse to customers or businesses, including the creation of shipping labels and packaging instructions.
- **Order Fulfillment**: For businesses using warehousing services, the CMS should allow orders to be fulfilled directly from the warehouse, generating packing lists and invoices.

5.Customer Support Integration

- **Helpdesk and Ticketing**: The system should provide a helpdesk or ticketing system where users can report issues, request support, or track the resolution of their queries.

Data Dictionary

organizations Table

Column Name	Data Type	Description
org_id	SERIAL	Primary Key, Auto-increment
org_name	VARCHAR(255)	Not Null, Organization name
org_email	VARCHAR(255)	Not Null, Unique, Organization email
org_contact	VARCHAR(15)	Not Null, Organization contact number
org_headOffice	TEXT	Not Null, Address of head office
org_password	TEXT	Not Null, Organization password
org_pincode	VARCHAR(10)	Not Null, Pincode of organization
org_manager_name	VARCHAR(255)	Not Null, Manager's name
manager_email	VARCHAR(255)	Not Null, Manager's email
manager_contact	VARCHAR(15)	Not Null, Manager's contact number

delivery_staff Table

Column Name	Data Type	Description
staff_id	VARCHAR(50)	Primary Key, Staff ID
org_name	VARCHAR(255)	Not Null, Organization name
staff_name	VARCHAR(100)	Not Null, Staff name
staff_city	VARCHAR(100)	Not Null, Staff city
staff_contact	VARCHAR(15)	Not Null, Staff contact number
staff_email	VARCHAR(100)	Not Null, Unique, Staff email
created_at	TIMESTAMP	Default: Current Timestamp

branches Table

Column Name	Data Type	Description
branch_id	VARCHAR(50)	Primary Key, Branch ID
org_name	VARCHAR(255)	Not Null, Organization name
branch_city	VARCHAR(100)	Not Null, Branch city
branch_state	VARCHAR(100)	Not Null, Branch state
branch_email	VARCHAR(100)	Not Null, Branch email
branch_contact	VARCHAR(10)	Not Null, Branch contact number

services Table

Column Name	Data Type	Description
service_id	VARCHAR(50)	Primary Key, Service ID
org_name	VARCHAR(255)	Not Null, Organization name
service_name	VARCHAR(255)	Not Null, Service name

warehouses Table

Column Name	Data Type	Description
warehouse_id	VARCHAR(50)	Primary Key, Warehouse ID
org_name	VARCHAR(255)	Not Null, Organization name
capacity	INT	Not Null, Warehouse capacity
state	VARCHAR(100)	Not Null, Warehouse state
city	VARCHAR(100)	Not Null, Warehouse city

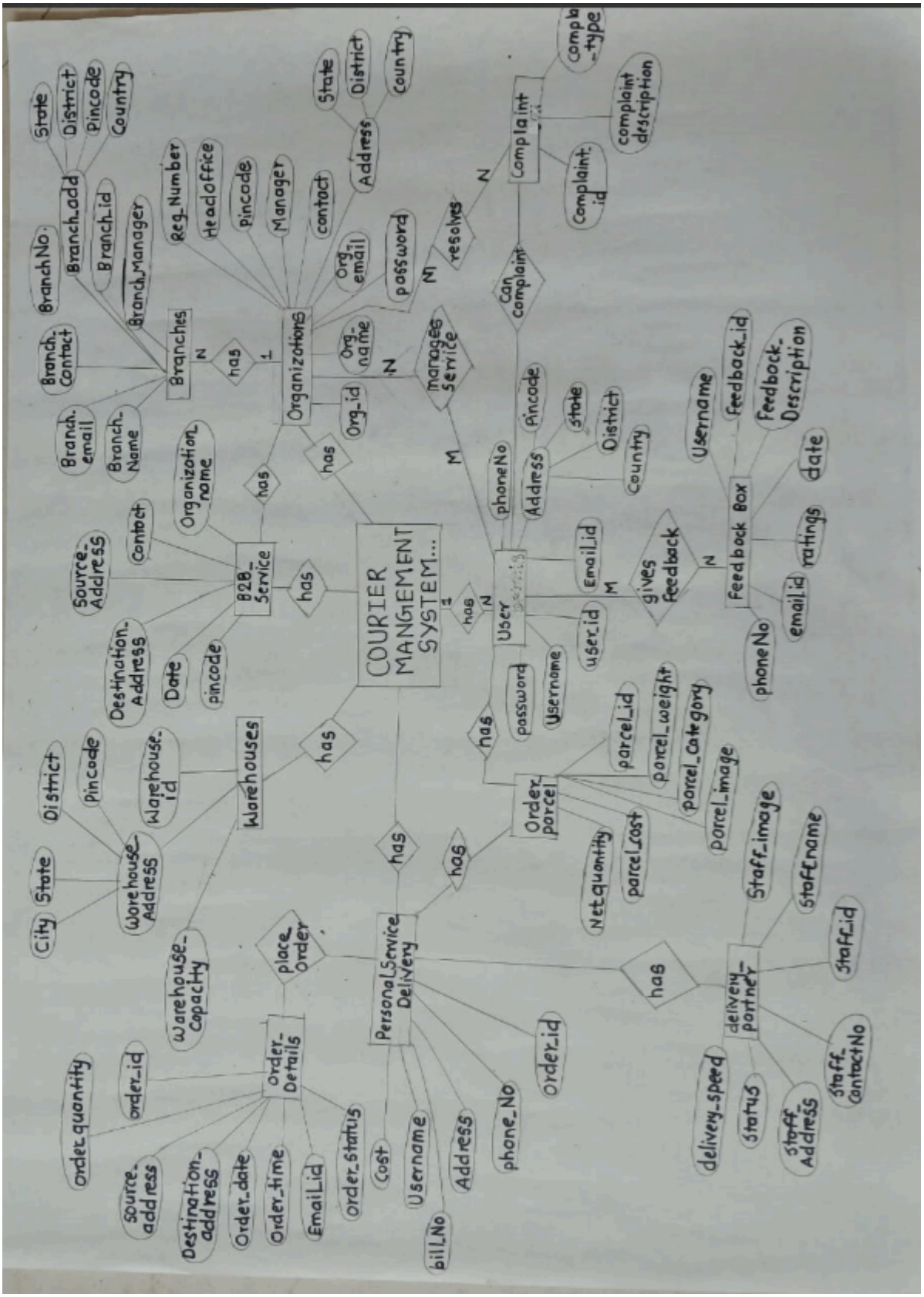
parcels Table

Column Name	Data Type	Description
parcel_id	VARCHAR(255)	Primary Key, Parcel ID
user_id	INT	Not Null, Foreign Key referencing `users(user_id)`
username	VARCHAR(255)	User's name
org_name	VARCHAR(255)	Not Null, Organization name
parcel_qty	INT	Not Null, Parcel quantity
parcel_cost	DECIMAL(10, 2)	Not Null, Parcel cost
parcel_description	TEXT	Not Null, Parcel description
warehouse_id	VARCHAR(255)	Not Null, Foreign Key referencing `warehouses(warehouse_id)`
delivery_staff	VARCHAR(255)	Not Null, Foreign Key referencing `delivery_staff(staff_id)`
order_date	DATE	Not Null, Date when parcel was ordered

user Table

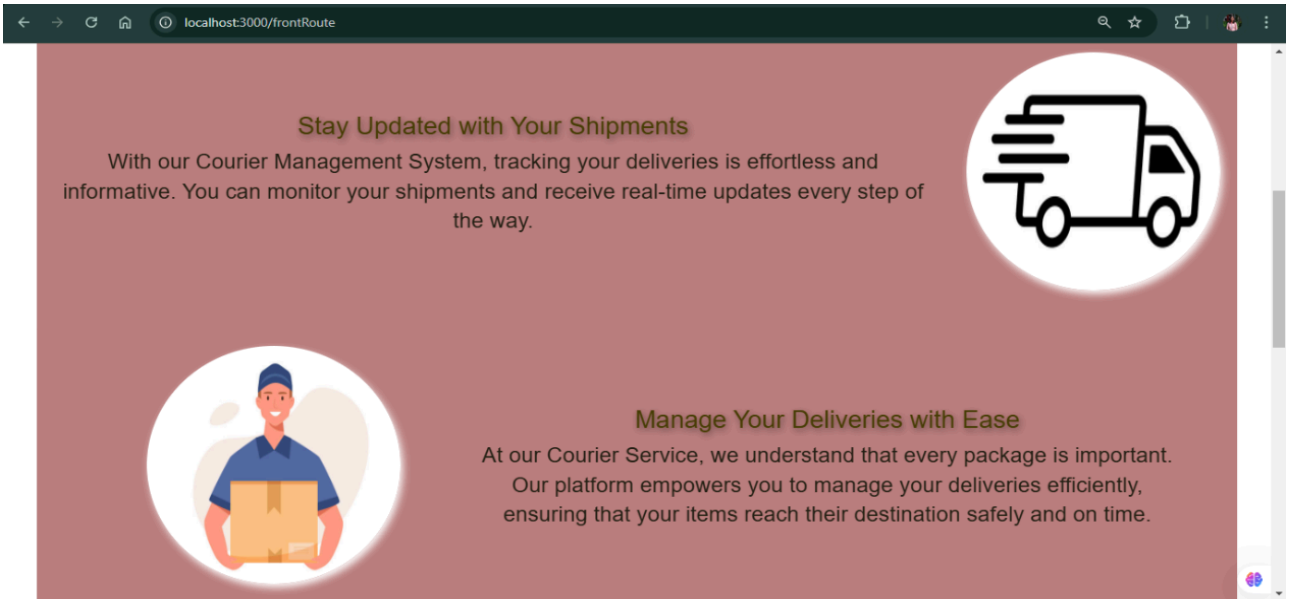
Column Name	Data Type	Description
id	INT	Primary Key, User ID
username	VARCHAR(50)	Not Null, User's username
password	VARCHAR(60)	Not Null, User's password
email	VARCHAR(100)	Not Null, User's email
city	VARCHAR(50)	User's city (optional)
phone	VARCHAR(15)	User's phone number (optional)

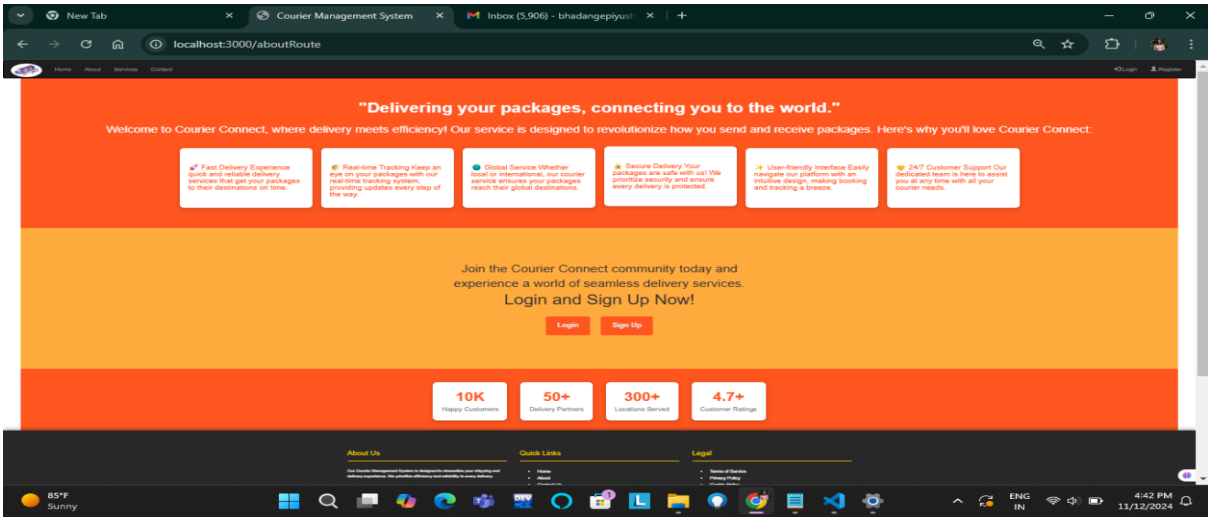
Relational schema

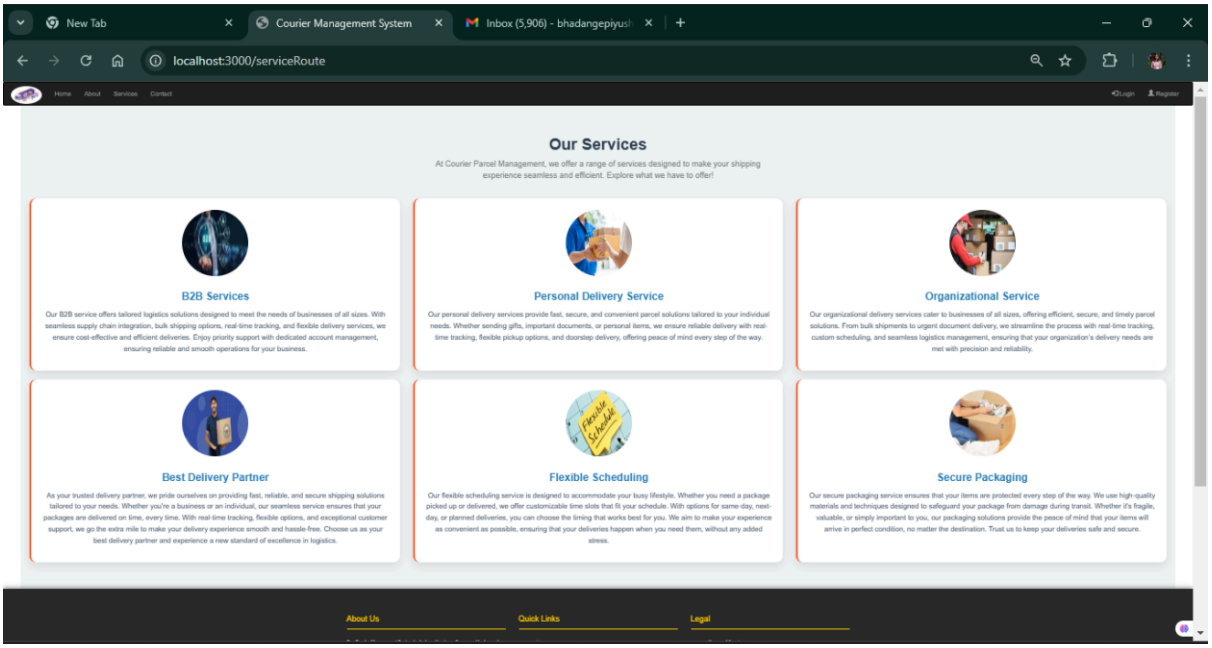


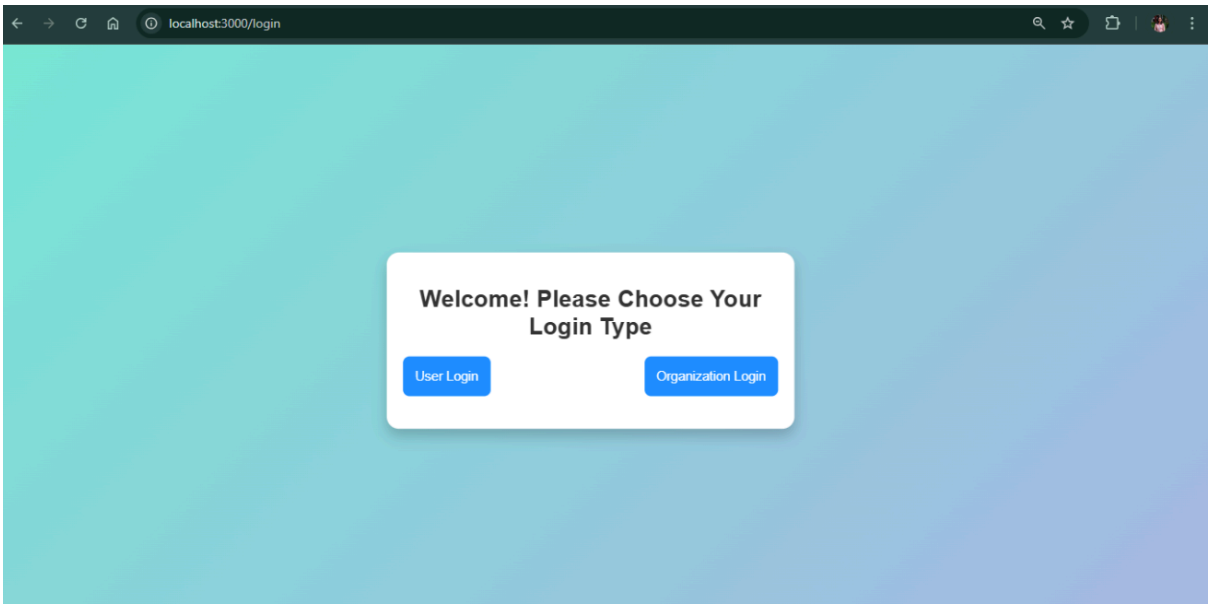
Graphical User Interface

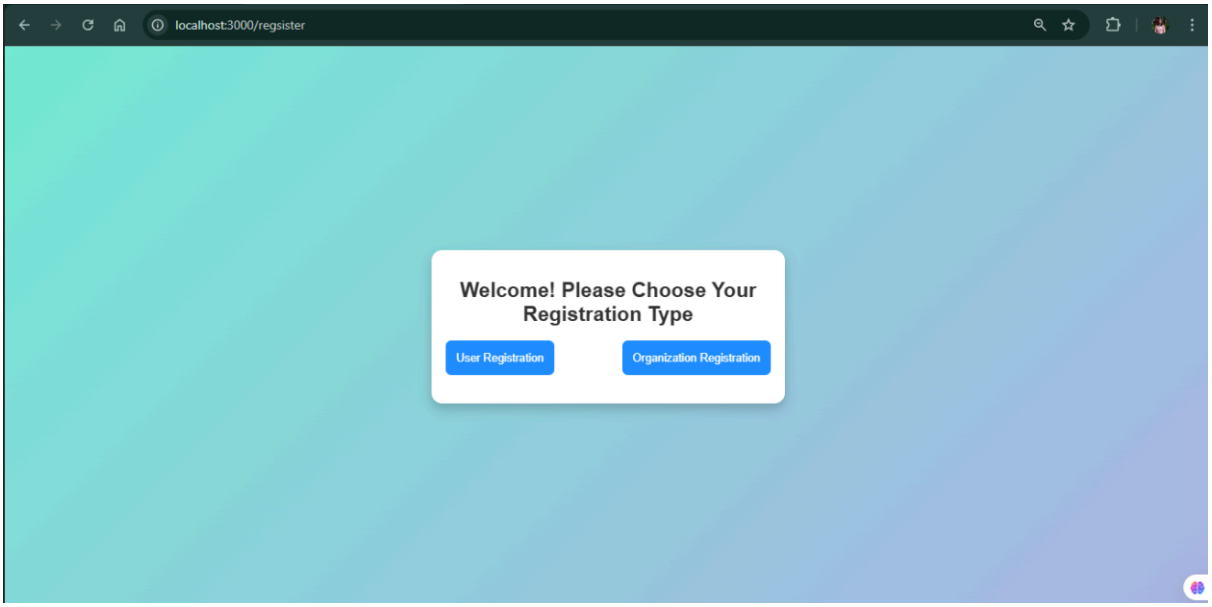
Home Page



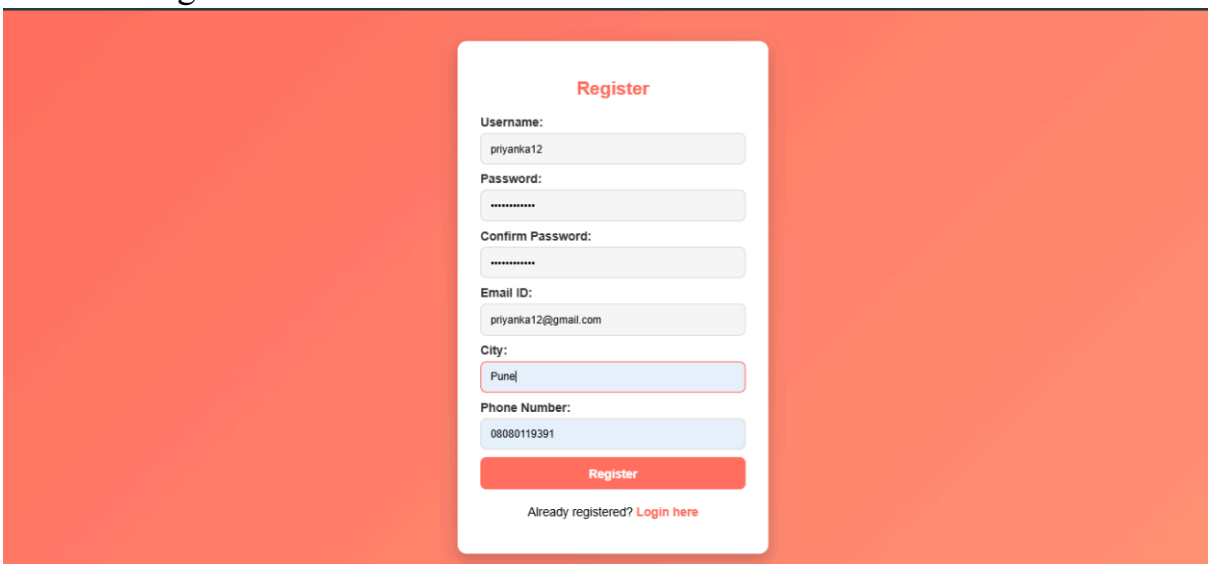

New Tab
Courier Management System
Inbox (5,906) - bhadangepiyush
localhost:3000/aboutRoute
"Delivering your packages, connecting you to the world."
Welcome to Courier Connect, where delivery meets efficiency! Our service is designed to revolutionize how you send and receive packages. Here's why you'll love Courier Connect:
Join the Courier Connect community today and experience a world of seamless delivery services.
Login and Sign Up Now!
Login
Sign Up
10K
Happy Customers
50+
Delivery Partners
300+
Locations Served
4.7+
Customer Ratings
About Us
Quick Links
Legal
85°F
Sunny
ENG
IN
4:42 PM
11/12/2024

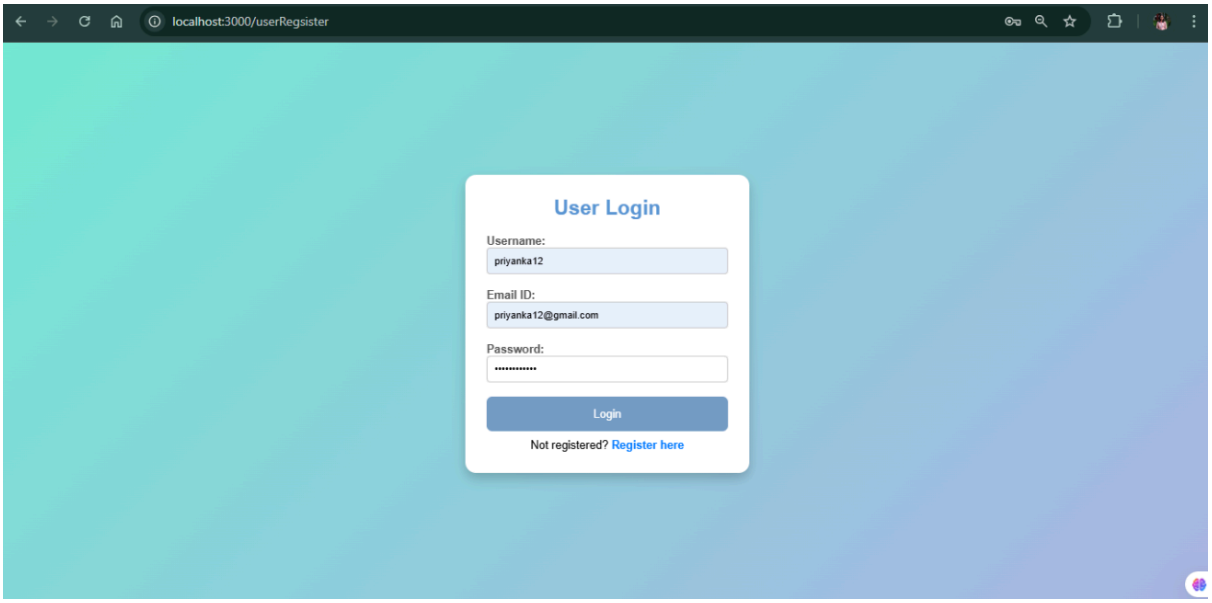

New Tab
Courier Management System
Inbox (5,906) - bhadangepiyush
localhost:3000/serviceRoute
Our Services
At Courier Parcel Management, we offer a range of services designed to make your shipping experience seamless and efficient. Explore what we have to offer!
B2B Services
Our B2B service offers tailored logistics solutions designed to meet the needs of businesses of all sizes. With seamless supply chain integration, bulk shipping options, real-time tracking, and flexible delivery services, we ensure cost-effective and efficient deliveries. Enjoy priority support with dedicated account management, ensuring reliable and smooth operations for your business.
Personal Delivery Service
Our personal delivery services provide fast, secure, and convenient parcel solutions tailored to your individual needs. Whether sending gifts, important documents, or personal items, we ensure reliable delivery with real-time tracking, flexible pickup options, and doorstep delivery, offering peace of mind every step of the way.
Organizational Service
Our organizational delivery services cater to businesses of all sizes, offering efficient, secure, and timely parcel solutions. From bulk shipments to urgent document delivery, we streamline the process with real-time tracking, custom scheduling, and seamless logistics management, ensuring that your organization's delivery needs are met with precision and reliability.
Best Delivery Partner
As your trusted delivery partner, we pride ourselves on providing fast, reliable, and secure shipping solutions tailored to your needs. Whether you're a business or an individual, our seamless service ensures that your packages are delivered on time, every time. With real-time tracking, flexible options, and exceptional customer support, we go the extra mile to make your delivery experience smooth and hassle-free. Choose us as your best delivery partner and experience a new standard of excellence in logistics.
Flexible Scheduling
Our flexible scheduling service is designed to accommodate your busy lifestyle. Whether you need a package picked up or delivered, we offer customizable time slots that fit your schedule. With options for same-day, next-day, or planned deliveries, you can choose the timing that works best for you. We aim to make your experience as convenient as possible, ensuring that your deliveries happen when you need them, without any added stress.
Secure Packaging
Our secure packaging service ensures that your items are protected every step of the way. We use high-quality materials and techniques designed to safeguard your package from damage during transit. Whether it's fragile, valuable, or simply important to you, our packaging solutions provide the peace of mind that your items will arrive in perfect condition, no matter the destination. Trust us to keep your deliveries safe and secure.
About Us
Quick Links
Legal

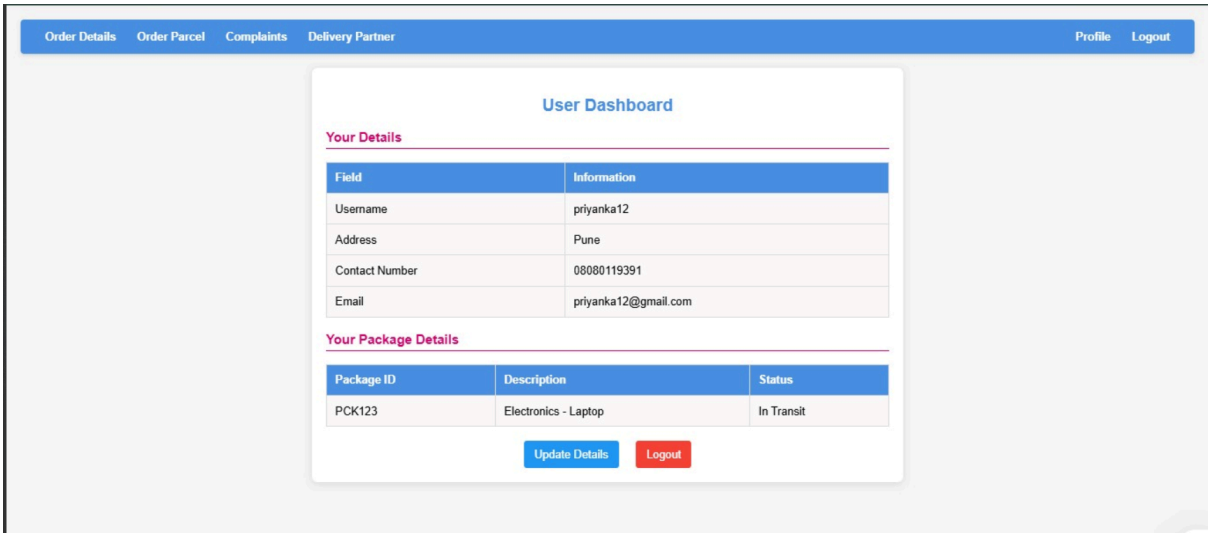

localhost:3000/login
Welcome! Please Choose Your Login Type
User Login
Organization Login

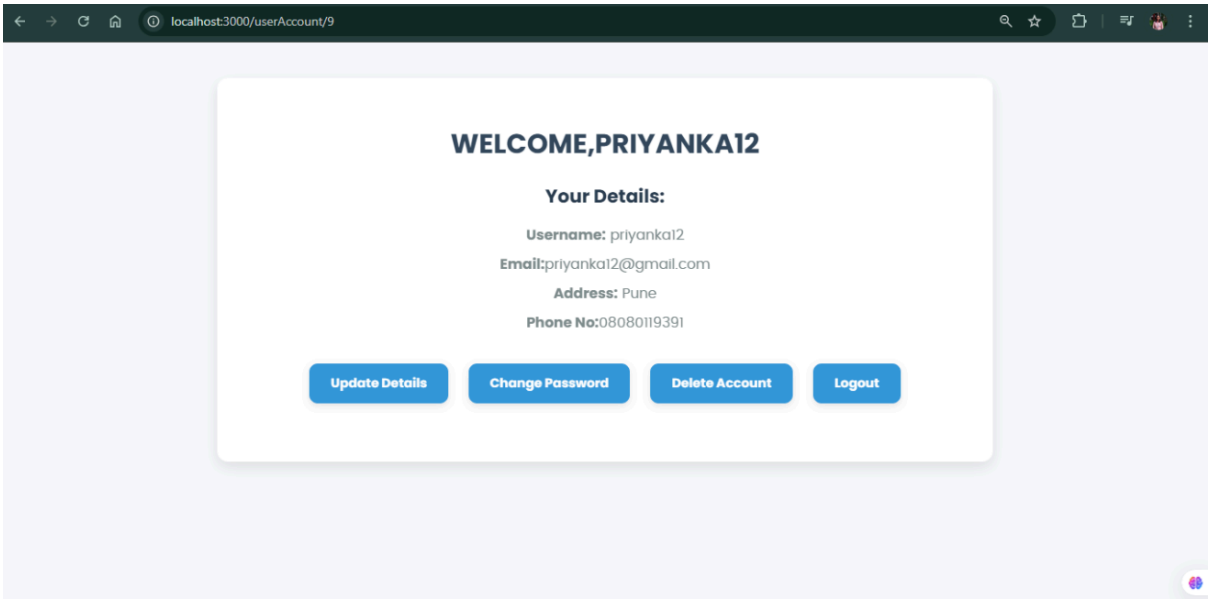


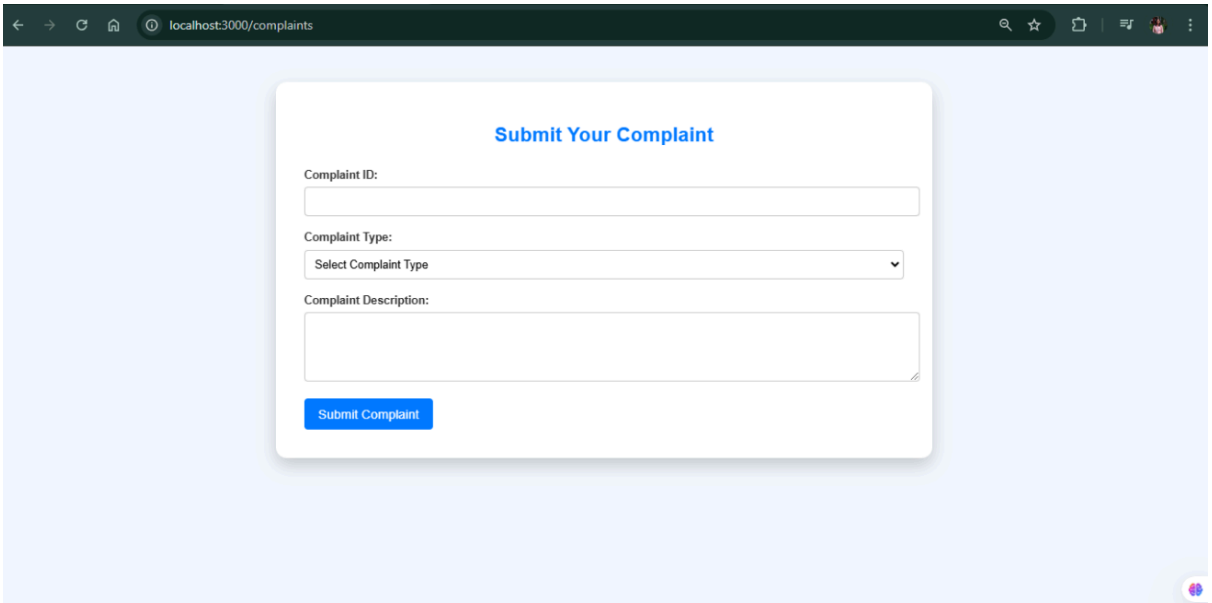
User side login

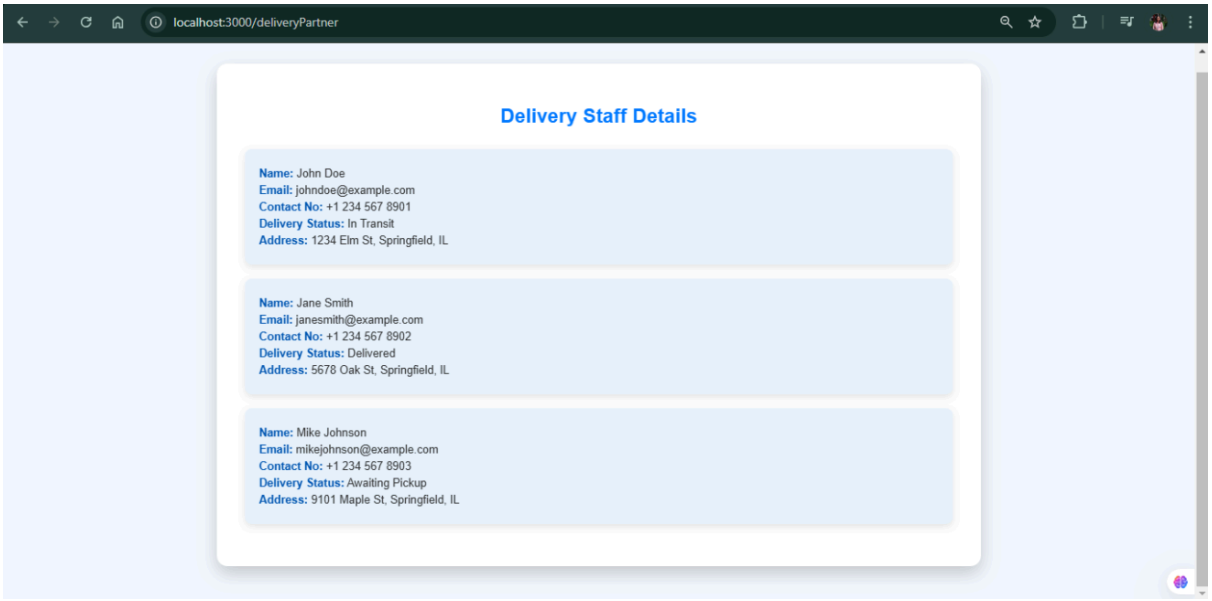




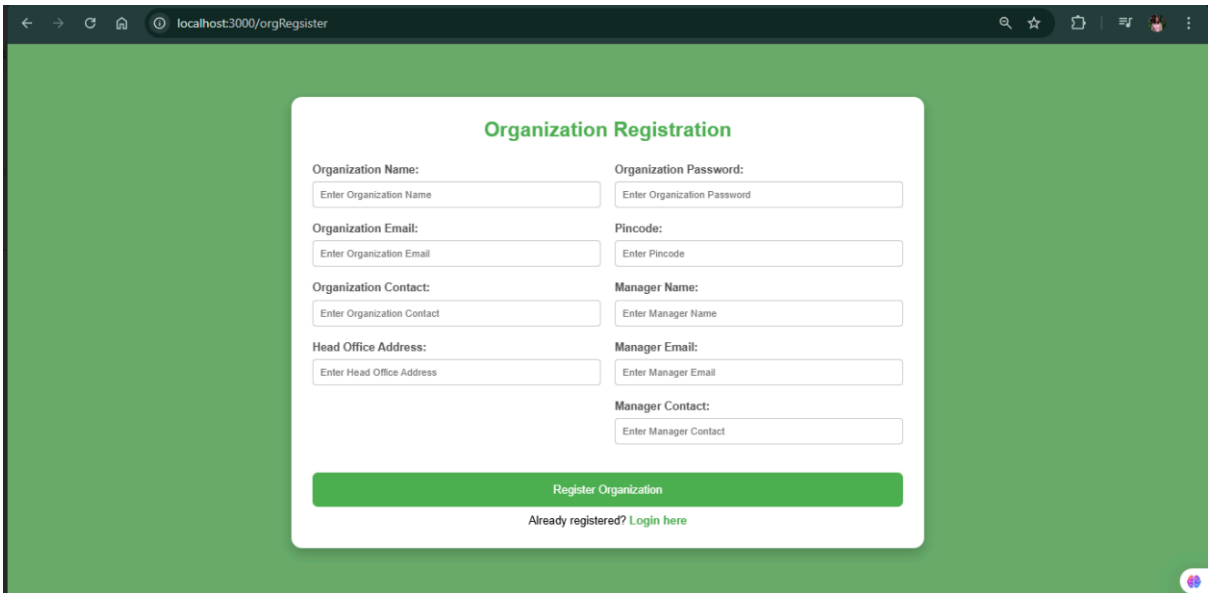

Order Details
Order Parcel
Complaints
Delivery Partner
Profile
Logout
User Dashboard
Your Details
Field
Information
Username
priyanka12
Address
Pune
Contact Number
08080119391
Email
priyanka12@gmail.com
Your Package Details
Package ID
Description
Status
PCK123
Electronics - Laptop
In Transit
Update Details
Logout

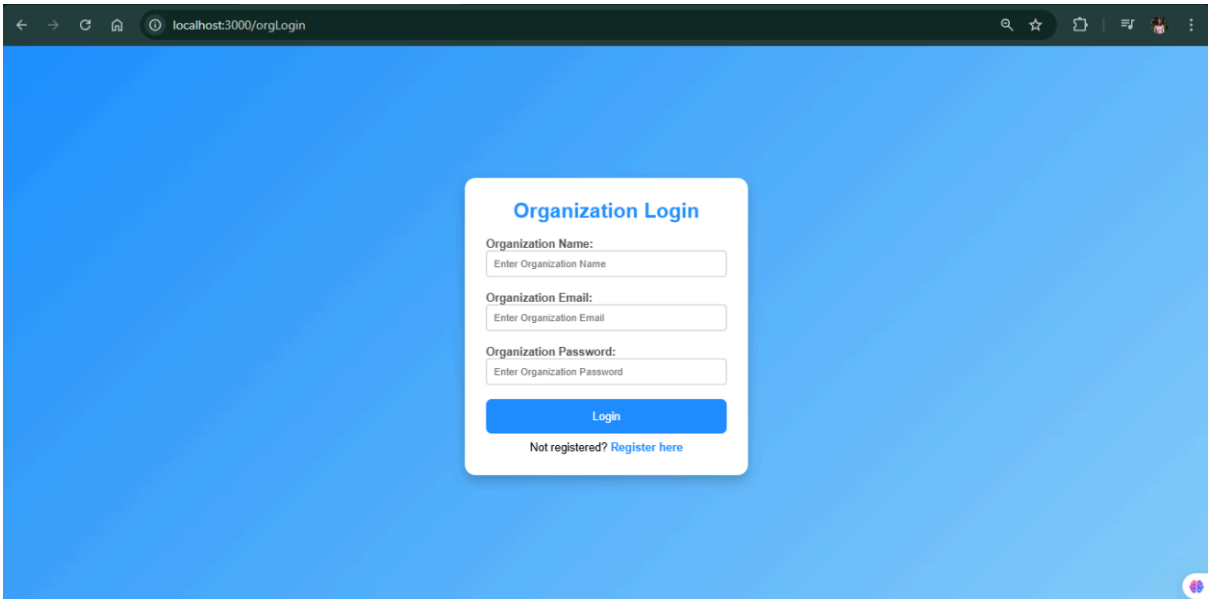

localhost:3000/userAccount/9
WELCOME,PRIYANKA12
Your Details:
Username: priyanka12
Email:priyanka12@gmail.com
Address: Pune
Phone No:08080119391
Update Details
Change Password
Delete Account
Logout

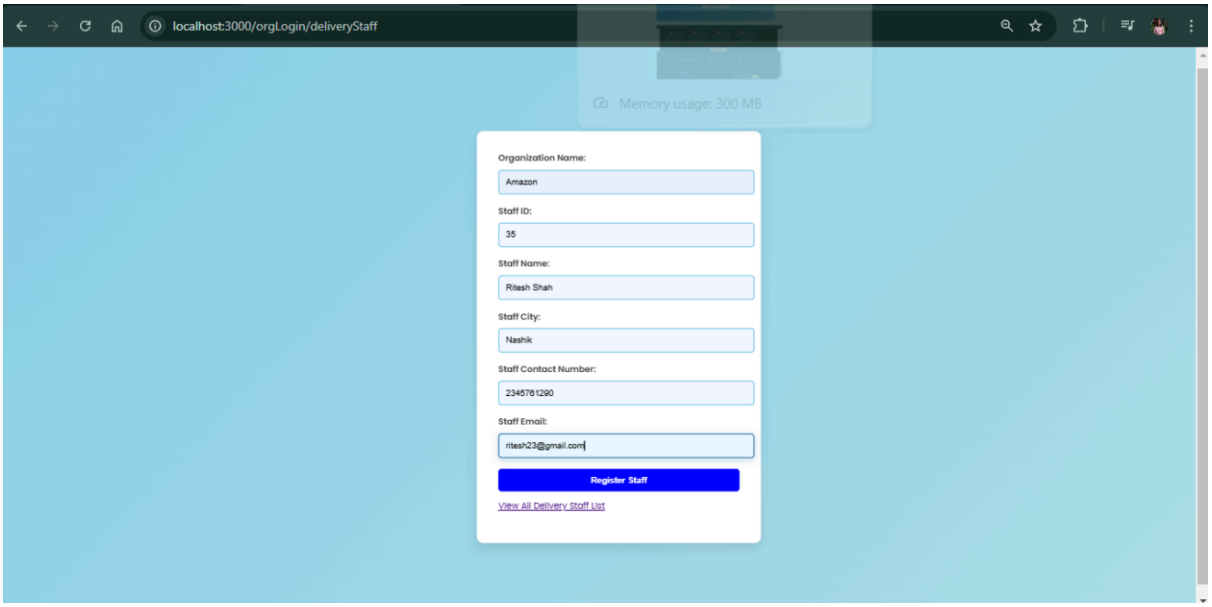

localhost:3000/complaints
Submit Your Complaint
Complaint ID:
Complaint Type:
Select Complaint Type
Complaint Description:
Submit Complaint



Organization login




localhost:3000/orgLogin
Organization Login
Organization Name:
Enter Organization Name
Organization Email:
Enter Organization Email
Organization Password:
Enter Organization Password
Login
Not registered? Register here

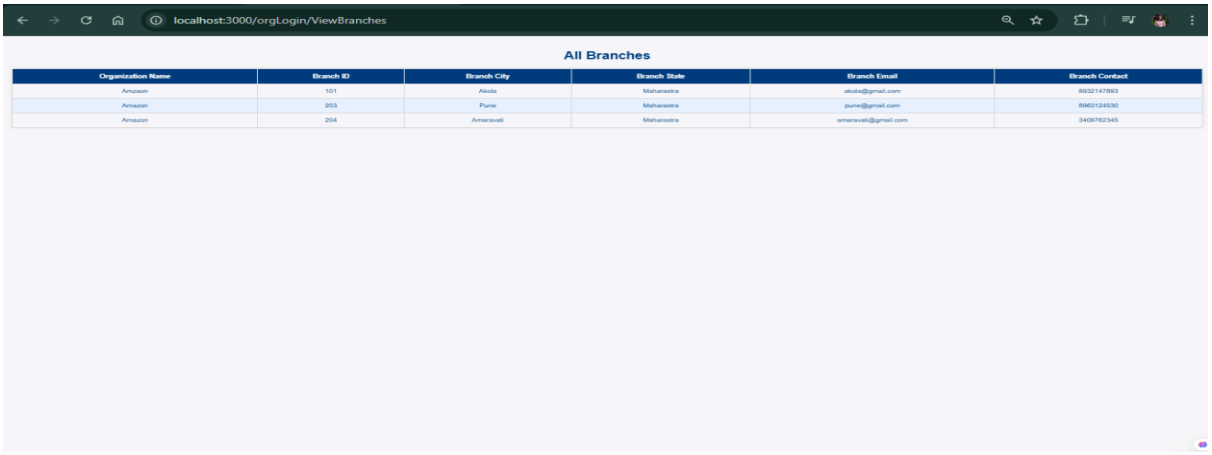

localhost:3000/orgLogin/deliveryStaff
Memory usage: 300 MB
Organization Name:
Amazon
Staff ID:
35
Staff Name:
Ritesh Shah
Staff City:
Nashik
Staff Contact Number:
2345761290
Staff Email:
ritesh23@gmail.com
Register Staff
View All Delivery Staff List

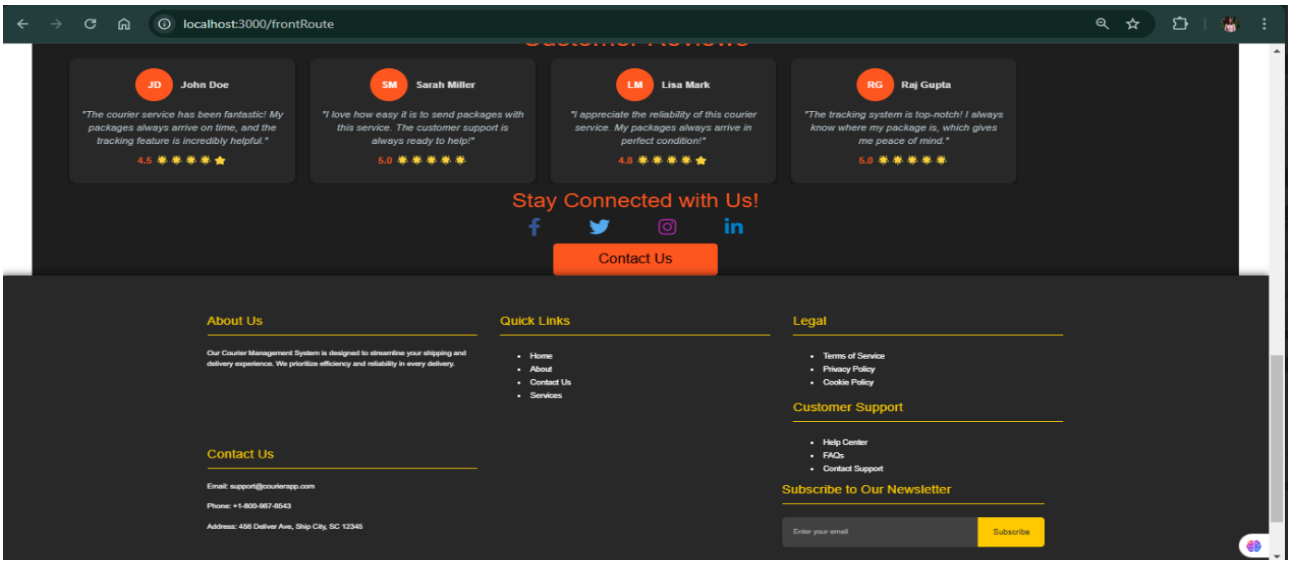
Warehouse


localhost:3000/orgLogin/viewWarehouses
All Warehouses
Organization Name
Warehouse ID
Capacity
State
City
Amzaon
501
50000
Maharashtra
Yevatmal
Amzaon
502
30000
Maharashtra
Nagpur
Amazon
503
10000000
Gujarat
Surat
Amazon
505
2000
Karnataka
Banglore

Delivery staff

Branches





Code Snippet

Index.js

```
const express=require("express");
const path=require("path");
const http=require("http");
const bodyParser=require("body-parser");

const app=express();
const port=3000 || process.env.port;
const server=http.createServer(app);

//routes
var frontRoute=require("./routes/frontRoute");
var aboutRoute=require("./routes/aboutRoute");
var serviceRoute=require("./routes/serviceRouter");
var userRegsister=require("./routes/userRegsister");
var orgRegsister=require("./routes/orgRegsister");
var userLogin=require("./routes/userLogin");
var orgLogin=require("./routes/orgLogin");
var login=require("./routes/login");
var regsister=require("./routes/regsister");


var userAccount=require("./routes/userAccount");
var orderDetails=require("./routes/order_details");
var orderParcel=require("./routes/order_parcel");
var complaints=require("./routes/complaints");
var deliveryPartner=require("./routes/deliveryPartner");

// Body parser middleware
app.use(bodyParser.urlencoded({ extended: false }));
app.use(bodyParser.json());

// Configure Express
app.set('view engine', 'ejs');
app.set('views', path.join(__dirname, 'views'));
app.use(express.static(path.join(__dirname, 'public')));

app.use("/frontRoute",frontRoute);
app.use("/aboutRoute",aboutRoute);
app.use("/serviceRoute",serviceRoute);
app.use("/userRegsister",userRegsister);
```

```
app.use("/orgRegsister",orgRegsister);
app.use("/userLogin",userLogin);
app.use("/orgLogin",orgLogin);
app.use("/login",login);
app.use("/regsister",regsister);

app.use("/userAccount",userAccount);
app.use("/orderDetails",orderDetails);
app.use("/orderParcel",orderParcel);
app.use("/complaints",complaints);
app.use("/deliveryPartner",deliveryPartner);

app.listen(port,()=>{
   console.log(server is running at ${port});
})
```

Organization login

```
//orgLogin.js
var express = require("express");
var router = express.Router();
var bodyParser = require("body-parser");
var path = require("path");
const {Pool}=require("pg");
const bcrypt = require("bcryptjs/dist/bcrypt");

const pool=new Pool({
   user: "postgres",
   host: "localhost",
   database: "NEW_CMS",
   password: "15Vijya2003@",
   port: 5432,
})

router.use(bodyParser.urlencoded({extended:true}));
pool.connect((err) => {
   if (err) {
      console.error('Connection error', err.stack);
   } else {
      console.log('Connected to database');
   }
});
router.get("/",(req,res)=>{
   res.render("orgLogin");
```

```
})

router.post("/",async(req,res)=>{
    const {org_name,org_email,org_password}=req.body;

    try{
        const query='SELECT * FROM organizations WHERE org_email = $1';
        const values=[org_email];

        const result=await pool.query(query,values);

        if(result.rows.length==0){
            return res.status(400).send("Organization not found. Please check your email
or register.");
        }

        const organization=result.rows[0];

        if(organization.org_name!=org_name){
            return res.status(400).send("Incorrect organization name.");
        }

        const isPasswordVaild=await
bcrypt.compare(org_password,organization.org_password);

        res.render("org_dashborad",{organization});
    }catch(error){
        console.log(error);
        res.status(500).send("An error occurred while logging in");
    }
})

router.get("/sendParcel",(req,res)=>{
    res.render("sendparcel");
})

router.post("/sendParcel",async(req,res)=>{
    const {
        userId,
        username,
        orgName,
        parcelId,
        parcelQty,
        parcelCost,
        parcelDescription,
        warehouseId,
```

```
        deliveryStaff,
        orderDate
    } = req.body;

    const query = `
    INSERT INTO parcels (user_id, username, org_name, parcel_id, parcel_qty,
parcel_cost, parcel_description, warehouse_id, delivery_staff, order_date)
    VALUES ($1, $2, $3, $4, $5, $6, $7, $8, $9, $10)`;

    try {
        await pool.query(query, [userId, username, orgName, parcelId, parcelQty,
parcelCost, parcelDescription, warehouseId, deliveryStaff, orderDate]);
        res.send('Parcel submitted successfully!');
    } catch (error) {
        console.error('Error executing query', error.stack);
        res.status(500).send('Error submitting parcel');
    }
})



router.get("/deliveryStaff",(req,res)=>{
    res.render("delivery_staff")
})


router.post('/deliveryStaff', async (req, res) => {
    const { orgName, staffId, staffName, staffCity, staffContact, staffEmail } =
req.body;

    const query = `
        INSERT INTO delivery_staff (org_name, staff_id, staff_name, staff_city,
staff_contact, staff_email)
        VALUES ($1, $2, $3, $4, $5, $6)
    `;

    try {
        await pool.query(query, [orgName, staffId, staffName, staffCity, staffContact,
staffEmail]);
        res.send('Staff registered successfully!');
    } catch (error) {
        console.error('Error executing query', error.stack);
        res.status(500).send('Error registering staff');
    }
});
```

```
router.get("/branches",(req,res)=>{
   res.render("branches");
})

router.post("/branches",async(req,res)=>{
   const { orgName, branchId, branchCity, branchState, branchEmail, branchContact
} = req.body;

   const query = `
   INSERT INTO branches (org_name, branch_id, branch_city, branch_state,
branch_email, branch_contact)
   VALUES ($1, $2, $3, $4, $5, $6)`;

   try {
      await pool.query(query, [orgName, branchId, branchCity, branchState,
branchEmail, branchContact]);
      res.send('Branch added successfully!');
   } catch (error) {
      console.error('Error executing query', error.stack);
      res.status(500).send('Error adding branch');
   }
})

router.get("/services",(req,res)=>{
   res.render("orgServices");
})
router.post("/services",async(req,res)=>{
   const { orgName, serviceId, serviceName } = req.body;

   const query = `
      INSERT INTO services (org_name, service_id, service_name)
      VALUES ($1, $2, $3)
   `;

   try {
      await pool.query(query, [orgName, serviceId, serviceName]);
      res.send('Service added successfully!');
   } catch (error) {
      console.error('Error executing query', error.stack);
      res.status(500).send('Error adding service');
   }
})


router.get("/warehouse",(req,res)=>{
   res.render("warehouses");
```

```
})
router.post("/warehouse",async(req,res)=>{
    const { orgName, warehouseId, capacity, state, city } = req.body;

    const query = `
        INSERT INTO warehouses (org_name, warehouse_id, capacity, state, city)
        VALUES ($1, $2, $3, $4, $5)
    `;

    try {
        await pool.query(query, [orgName, warehouseId, capacity, state, city]);
        res.send('Warehouse added successfully!');
    } catch (error) {
        console.error('Error executing query', error.stack);
        res.status(500).send('Error adding warehouse');
    }
})

router.get("/logout",async(req,res)=>{
    res.render("orgLogin");
});

router.get("/ViewAllStaff",async(req,res)=>{
    const query = `
    SELECT org_name, staff_id, staff_name, staff_city, staff_contact, staff_email
    FROM delivery_staff`;
    try {
        const result = await pool.query(query);
        res.render('viewAllStaff', { staffList: result.rows });
    } catch (error) {
        console.error('Error fetching staff list', error.stack);
        res.status(500).send('Error fetching staff list');
    }
});

router.get("/viewWarehouses", async (req, res) => {
    const query = SELECT * FROM warehouses;

    try {
        const result = await pool.query(query);
        const warehouses = result.rows;
        res.render("viewWarehouses", { warehouses });
    } catch (error) {
        console.error('Error retrieving warehouses:', error);
        res.status(500).send('Error retrieving warehouses');
    }
```

```
});
router.get("/viewServices", async (req, res) => {
   const query = SELECT * FROM services;

   try {
      const result = await pool.query(query);
      const services = result.rows;
      res.render("viewServices", { services });
   } catch (error) {
      console.error('Error retrieving services:', error);
      res.status(500).send('Error retrieving services');
   }
});
router.get("/viewBranches", async (req, res) => {
   const query = SELECT * FROM branches;

   try {
      const result = await pool.query(query);
      const branches = result.rows;
      res.render("viewBranches", { branches });
   } catch (error) {
      console.error('Error retrieving branches:', error);
      res.status(500).send('Error retrieving branches');
   }
});
router.get("/ViewSendParcel", async (req, res) => {
   const query = SELECT * FROM parcels;

   try {
      const result = await pool.query(query);
      const parcels = result.rows; // Assuming PostgreSQL is being used
      res.render("viewSendParcel", { parcels });
   } catch (error) {
      console.error("Error retrieving parcel history:", error);
      res.status(500).send("Error retrieving parcel history");
   }
});
module.exports = router;
```

Organization Register

```
//orgRegsister.js
var express = require("express");
var router = express.Router();
var bodyParser = require("body-parser");
var path = require("path");
```

```
const {Pool}=require("pg");
const bcrypt = require("bcryptjs/dist/bcrypt");

const pool=new Pool({
    user: "postgres",
    host: "localhost",
    database: "NEW_CMS",
    password: "15Vijya2003@",
    port: 5432,
})
pool.connect((err) => {
    if (err) {
        console.error('Connection error', err.stack);
    } else {
        console.log('Connected to database');
    }
});

router.use(bodyParser.urlencoded({extended:true}));

router.get("/",(req,res)=>{
    res.render("orgRegsister");
})
router.post("/",async(req,res)=>{
    const{
        org_name,
        org_email,
        org_contact,
        org_headOffice,
        org_password,
        org_pincode,
        org_manager_name,
        manager_email,
        manager_contact
    }=req.body;
    try{
        const checkEmailQuery='SELECT * FROM  organizations  WHERE
org_email=$1';
        const checkEmailValues=[org_email];

        const checkEmailResult=await
pool.query(checkEmailQuery,checkEmailValues);

        if(checkEmailResult.rows.length > 0){
            return res.status(400).send("THis email is already regsitered by another
organization");
```

```
        }

        const hashedPassword=await bcrypt.hash(org_password,10);

        const insertQuery=`
            INSERT INTO  organizations
            (org_name, org_email, org_contact, org_headOffice, org_password,
org_pincode, org_manager_name, manager_email, manager_contact)
            VALUES ($1, $2, $3, $4, $5, $6, $7, $8, $9) RETURNING org_id`;

        const insertValues=[org_name, org_email, org_contact, org_headOffice,
hashedPassword, org_pincode, org_manager_name, manager_email,
manager_contact];

        const result=await pool.query(insertQuery,insertValues);

        res.redirect("orgLogin");
    }catch(error){
        console.log(error);
        res.status(500).send("An error occurred during registration");;
    }
})


module.exports = router;
```

User Account

```
//userAccount.js
var express = require("express");
var router = express.Router();
var bodyParser = require("body-parser");
var path = require("path");
const {Pool}=require("pg");
const bcrypt = require("bcryptjs/dist/bcrypt");



const pool=new Pool({
    user: "postgres",
    host: "localhost",
    database: "NEW_CMS",
    password: "15Vijya2003@",
    port: 5432,
})
```

```
pool.connect((err) => {
    if (err) {
        console.error('Connection error', err.stack);
    } else {
        console.log('Connected to database');
    }
});

router.use(bodyParser.urlencoded({extended:true}));



router.get("/:user_id",async(req,res)=>{
    const userId=req.params.user_id;

    try{
        const Query='SELECT * FROM users WHERE  USER_ID =$1';
        const values=[userId];

        const result=await pool.query(Query,values);

        if(result.rows.length==0){
            return res.status(404).send("User not found.");
        }

        const user=result.rows[0];

         res.render("Account",{user});
    }catch(error){
        console.error("Error fetching user details: ", error.message);
        res.status(500).send("Internal Server Error");
    }
})

//GET METHOD OF UPDATE DETAILS
router.get("/update-details/:user_id",async(req,res)=>{
    const user_id=req.params.user_id;

    try{
        const query=SELECT * FROM users WHERE user_id =$1;

        const values=[user_id];

         const result=await pool.query(query,values);
```

```
        if(result.rows.length===0){
          return res.status(404).send("User not found");
        }

        const user=result.rows[0];

        res.render("updateUserDetails",{user});
    }catch(error){
        console.error("Error fetching user details: ", error.message);
    }
})

//GET METHOD FOR DELETING THE ACCOUNT
router.get("/delete-account/:user_id",async(req,res)=>{
    const user_id=req.params.user_id;

    try{
        const query=SELECT * FROM users WHERE user_id=$1;
        const values=[user_id];

        const result=await pool.query(query,values);

        if(result.rows.length==0){
            return res.status(400).send("User does not found");
        }

        const user=result.rows[0];
        res.render("deleteAccount",{user});
    }catch(error){
        console.error("Error in fetching the deatils",error.message);
    }
})
//GET METHOD IF CHANGE PASSWORD
router.get("/change-password/:user_id",async(req,res)=>{
    const user_id=req.params.user_id;

    try{
        const query=SELECT * FROM users WHERE user_id=$1;

        const values=[user_id];

        const result=await pool.query(query,values);

        if(result.rows.length==0){
            return res.status(400).send("User does not found");
        }
```

```
        const user=result.rows[0];

        res.render("changePassword",{user});
    }catch(error){
        console.error("Error fetching user details.",error.message);
    }
})

//POST METHOD OF UPDATE DEATILS
router.post("/update-details/:user_id",async(req,res)=>{
    const user_id=req.params.user_id;

    const {username,email_id,city,phone_no}=req.body;

    try{
        const query=UPDATE users SET username=$1 , email_id=$2 , city=$3 ,
phone_no =$4 WHERE user_id =$5;

        const values=[username,email_id,city,phone_no,user_id];

        await pool.query(query,values);
        res.redirect(/userAccount/${user_id});
    }catch(error){
        console.log("Error upating the user deatils",error.message);
        res.status(500).send("Internal Server Error");
    }
})


//POST METHOD OF CHANGE PASSWORD
router.post("/change-password/:user_id",async(req,res)=>{
    const user_id=req.params.user_id;
    const current_password=req.body.current_password;
    const new_password=req.body.new_password;
    const confirm_password=req.body.confirm_password;

    if(new_password!== confirm_password){
        req.send("Password desnot match");
    }

    try{
        const Query='SELECT * FROM  users WHERE user_id = $1';
        const values=[user_id];

        const result=await pool.query(Query,values);
```

```
        if(result.rows.length==0){
            return res.status(400).send("Users not found");
        }

        const user=result.rows[0];

        const isMatch=await bcrypt.compare(current_password,user.user_password);

        if(!isMatch){
            return res.status(400).send("Current password is incorrect.");
        }

        const hashedPassword= await bcrypt.hash(new_password,10);
        const updatedQuery='UPDATE users SET user_password=$1 WHERE
user_id=$2';
        const updateValues=[hashedPassword,user_id];

         await pool.query(updatedQuery,updateValues);

         return res.send("Password changes successfully!!");
    }catch(error){
        console.error("Error changing password: ", error.message);
    }
})

//POST REQUEST FOR DELETING THE FORM


router.post('/delete-account/:user_id', async (req, res) => {
    const user_id = req.params.user_id;
    const { password, confirmDelete } = req.body;

    // Check if user confirmed account deletion
    if (!confirmDelete) {
        return res.status(400).send("You must confirm that you want to delete the
account.");
    }

    try {
        // Fetch user details from database
        const query = SELECT * FROM users WHERE user_id = $1;
        const value = [user_id];

        const result = await pool.query(query, value);
```

```
        // Check if the user exists
        if (result.rows.length === 0) {
            return res.status(404).send("User not found.");
        }

        const user = result.rows[0];

        // Compare entered password with the hashed password
        const hashedPassword = user.user_password;
        const isPasswordValid = await bcrypt.compare(password, hashedPassword);

        // If password is invalid, send error response
        if (!isPasswordValid) {
            return res.status(400).send("Incorrect password.");
        }

        // Delete user from database
        const deleteQuery = 'DELETE FROM users WHERE user_id = $1';
        const deleteValues = [user_id];
        await pool.query(deleteQuery, deleteValues);

        // Send success message
        res.render("userLogin");
    } catch (error) {
        console.error(error);
        res.status(500).send("An error occurred.");
    }
});



module.exports = router;
```

User Login

```
//userLogin.js
var express = require("express");
var router = express.Router();
var bodyParser = require("body-parser");
var path = require("path");
const {Pool}=require("pg");
const bcrypt = require("bcryptjs/dist/bcrypt");
var jwt=require("jsonwebtoken");

const pool=new Pool({
```

```
    user: "postgres",
    host: "localhost",
    database: "NEW_CMS",
    password: "15Vijya2003@",
    port: 5432,
})

pool.connect((err) => {
    if (err) {
        console.error('Connection error', err.stack);
    } else {
        console.log('Connected to database');
    }
});

router.use(bodyParser.urlencoded({extended:true}));

if (typeof localStorage === "undefined" || localStorage === null) {
    const LocalStorage = require('node-localstorage').LocalStorage;
    localStorage = new LocalStorage('./scratch');
}

function checkLogin(req,res,next){
    var mytoken=localStorage.getItem('mytoken');
    try {
      //it will verify mytoken that we have created when we are go to login page
      jwt.verify(mytoken, 'loginToken');
    } catch(err) {
      res.send("You need to login to access courier parcel management");
    }
    //if login verify then it will pass to next method and display Employee record if its
verify.
    next();
}


router.get("/",(req,res)=>{
    res.render("userLogin");
})
router.get("/",function(req,res,next){
    res.render("userLogin");
})
router.post("/",async(req,res)=>{
    const username=req.body.username;
    const email_id=req.body.email_id;
    const password=req.body.password;
```

```
    try{
        const query='SELECT * FROM users WHERE username=$1 AND
email_id=$2';

        const values=[username,email_id];

        const result=await pool.query(query,values);

        if(result.rows.length==0){
            return res.status(400).send("Invalid username or email.");
        }

        const user=result.rows[0];

        const isMatch =await bcrypt.compare(password,user.user_password);

        if(!isMatch){
            return res.status(400).send("Invalid username or password.");
        }



        //jwt for creating login token
        var token = jwt.sign({ foo: 'bar' }, 'loginToken');
        localStorage.setItem('mytoken', token);

        res.render("userdashborad",{user});
    }catch(error){
        console.error("Error during login: ", error.message);
        res.status(500).send("Internal Server Error");
    }
})


router.get("/logout", function (req, res, next){
    //this will remove mytoken form local Storage.
    localStorage.removeItem('mytoken');
    // res.send("Logout Successfully.");
    res.render("userLogin");
  });



module.exports = router;
```

User Register

```
//userRegsister.js
var express = require("express");
var router = express.Router();
var bodyParser = require("body-parser");
var path = require("path");
const {Pool}=require("pg");
const bcrypt = require('bcryptjs');

const pool=new Pool({
    user: "postgres",
    host: "localhost",
    database: "NEW_CMS",
    password: "15Vijya2003@",
    port: 5432,
})
pool.connect((err) => {
    if (err) {
        console.error('Connection error', err.stack);
    } else {
        console.log('Connected to database');
    }
});
router.use(bodyParser.urlencoded({extended:true}));

router.get("/",(req,res)=>{
    res.render("userRegsister");
})


router.post("/", async (req, res) => {
    const { username, password, email_id, city, phone_number } = req.body;

    const confirmPassword = req.body['confirm_password'];
    if (password !== confirmPassword) {
        return res.status(400).send("Confirm Password does not match the Password.");
    }

    try {
        // Check if username or email already exists
        const checkUserQuery = 'SELECT * FROM users WHERE username = $1 OR
email_id = $2';
        const checkUsersValues = [username, email_id];
```

```
        const result = await pool.query(checkUserQuery, checkUsersValues);

        if (result.rows.length > 0) {
            return res.status(400).send("Username or Email is already registered! Try with
another email.");
        }

        // Hash the password asynchronously
        const hashedPassword = await bcrypt.hash(password, 10);

        // Insert the user into the database
        const insertUserQuery = INSERT INTO users (username, user_password,
email_id, city, phone_no) VALUES ($1, $2, $3, $4, $5);
        const insertUserValues = [username, hashedPassword, email_id, city,
phone_number];

        await pool.query(insertUserQuery, insertUserValues);

        //res.send("User registered successfully!");
        res.render("userLogin");

    } catch (error) { // Catching the error and logging it
        console.error("Error in registration of Users: ", error.message);
        res.status(500).send("Internal Server Error");
    }
});

module.exports = router;
```

Conclusion

The Courier Management System efficiently streamlines courier operations, handling user, organization, staff, and parcel management. It ensures smooth parcel flow through structured data relationships, improving tracking and delivery accuracy. With a focus on data integrity and efficient resource management, the system enables organizations to enhance service delivery, reduce errors, and boost customer satisfaction. This system provides scalability and flexibility to accommodate business growth, ultimately supporting the success of the courier service.