COURIER MANAGEMENT SYSTEM

Candidate Name: MANTHI BINDU MADHAVI

Regd.No: 2341111

COURIER MANAGEMENT SYSTEM

Report Submitted to

D.N.R. COLLEGE, P.G.COURSES (AUTONOMOUS)

for the award of the degree

of

MASTER OF COMPUTER APPLICATIONS

by

MANTHI BINDU MADHAVI

REGD. No: 2341111

Under the Guidance of

Sri. A. NAGA RAJU, M.C.A

Assistant Professor

DEPARTMENT OF COMPUTER APPLICATIONS

D.N.R.COLLEGE, P.G.COURSES (AUTONOMOUS)

(Affiliated to ADIKAVI NANNAYA UNIVERSITY)

BHIMAVARAM-534202

APRIL-2025

DEPARTMENT OF COMPUTER APPLICATIONS

D.N.R.COLLEGE, P.G.COURSES (AUTONOMOUS)

(Affiliated to ADIKAVI NANNAYA UNIVERSITY)

BHIMAVARAM-534202

CERTIFICATE

This is to certify that the project Report entitled, "**COURIER MANAGEMENT SYSTEM**" submitted by **Mr.** "**MANTHI BINDU MADHAVI**" Redgno: **2341111** to **"D.N.R. College, P.G. Courses (Autonomous)"**, affiliated to "**Adikavi Nannaya University"**, Rajahmahendravaram, Andhra Pradesh, India, is a record of Bonafide Project work carried out by him under my supervision and guidance and is worthy of consideration for the award of the degree of Master of Computer Applications.

Date:

Internal Guide	**Head of the Department**
Mr. A. Naga Raju, M.C.A	**Sri ,k. Rambabu, M.TECH**

External Examiner	**Internal Examiner**

DECLARATION

I certify that

a. The work contained in this report is original and has been done by me under the guidance of my supervisor(s).

b. The work has not been submitted to any other Institute for any degree or diploma.

c. I have followed the guidelines provided by the university in preparing the report.

d. I have conformed to the norms and guidelines given in the Ethical Code of Conduct of the Institute.

e. Whenever I have used materials (data, theoretical analysis, figures, and text) from other sources, I have given due credit to them by citing them in the text of the report and giving their details in the references. Further, I have taken permission from the copyright owners of the sources, whenever necessary.

MANTHI BINDU MADHAVI

Regd.No: 2341111

ACKNOWLEDGEMENT

I would like to take the privilege of the opportunity to express my gratitude into Project work of "**COURIER MANAGEMENT SYSTEM**" enabled me to express my thanks to **Mr. A.NAGA RAJU, Assistant Professor**, Department of Computer Applications, D.N.R College, P.G.Courses, for guidance since the project started. I feel that it is a great privilege of completing my project under his guidance.

I owe my gratitude to our beloved **SRI. K. RAMBAB**U Head of the Department, Department of Computer Applications, D.N.R College, P.G.Courses , for being a source of inspiriting and encouraging me in successful completion of the project work.

I am thankful to our Honorable Principal, D.N.R College, P.G.Courses , Sri. ,Who has shown keen interest and encouraged me by providing all the facilities to complete my project successful.

I wish to express my sincere thanks to our Management by providing all the facilities and infrastructure to complete my project successfully and I wish to express my sincere thanks to all Teaching and Non-Teaching staff of Department of Computer Applications, D.N.R College, P.G.Courses.

I am very thankful to all my friends and my family members who had given me good co-operation and Suggestion throughout this project and helped me in successful completion.

MANTHI BINDU MADHAVI

Regd.No: 2341111

COURIER MANAGEMENT SYSTEM

ABSTRACT

ABSTRACT

This Django-based application aims to streamline courier management by integrating functionalities such as parcel tracking, feedback collection, employee management, and courier status updates. The system uses MySQL as the backend database and enables users to track the status of their couriers in real-time by interacting with the system through web interfaces. The feedback mechanism provides valuable insights into the service quality through a user-friendly display of feedback data, including a pie chart visualization of feedback rankings. Employees and admins can update parcel status and track courier movement on a dynamic map. The system also supports file uploads for images associated with parcels, leveraging Django's file handling capabilities. Key features include parcel tracking with dynamic updates, interactive map integration, courier feedback collection, and an employee login system for secure access to administrative actions. The application provides a comprehensive approach to courier service management, offering both user-friendly interfaces and efficient backend processes for tracking, status updates, and feedback analysis.

INDEX

TITLE **PAGE NO**

LIST OF DIAGRAMS

INTRODUCTION

1.INTRODUCTION

In today’s fast-paced and technology-driven world, logistics and courier services play a crucial role in global connectivity. With the exponential growth of e-commerce, the demand for efficient and real-time courier tracking systems has become paramount. Customers expect transparent, reliable, and prompt delivery services, and businesses seek ways to streamline their operations to meet these expectations. In this context, an automated **Courier Management System** built using Django, a high-level Python web framework, proves to be an essential solution that bridges the gap between service providers and consumers.

The project titled **“Courier Management System using Django”** is designed to enhance the efficiency of courier service tracking, status management, and customer interaction. This system not only helps customers to track their parcels but also enables employees to update parcel statuses in real-time, submit feedback, and collect shipment data in a structured manner. The system provides a user-friendly web interface that supports multiple roles such as employees, administrators, and customers, each with tailored functionalities to meet their needs.

The core idea of this project revolves around the integration of modern web development techniques with robust backend logic to ensure real-time data processing and visualization. Developed using Django, the system incorporates powerful Python libraries such as pymysql for database connectivity, matplotlib for dynamic feedback chart generation, and numpy for statistical data handling. The use of Google Maps embedding further enhances the usability by allowing users to visualize current parcel locations directly on a map interface.

Motivation

Manual tracking and record-keeping in courier services often result in lost or misplaced parcels, delivery delays, and a lack of transparency. Such issues can diminish customer trust and harm business reputations. The motivation behind this project is to eliminate these inefficiencies by automating the entire courier management process. From parcel collection and delivery updates to feedback visualization and map-based location tracking, every component is streamlined for clarity and efficiency.

A centralized digital platform helps employees update delivery progress without physical paperwork, allows customers to stay informed about their shipments, and equips administrators with comprehensive data to make informed decisions. The feedback and performance analytics provide insights into customer satisfaction and highlight areas of improvement. Overall, the system promotes accountability, responsiveness, and a seamless delivery experience.

Key Features

This Courier Management System provides the following key features:

Courier Collection Interface: Employees can log collected courier details including sender and receiver information, delivery date, parcel description, weight, amount, and upload images of parcels.

Parcel Status Updates: Employees are allowed to update the current location, status (e.g., "In Transit", "Delivered"), and date of status change. This ensures that customers can track their shipments in real-time.

Courier Tracking by Customers: End-users can track their parcels using parcel ID, view delivery information, current location, and the parcel image, and even locate the current position on an embedded Google Map.

Feedback System: Customers can submit feedback and rank their courier experience. The system stores these entries and visualizes the feedback distribution using a pie chart for administrative review.

Feedback Visualization: Using matplotlib, the system generates a dynamic pie chart that shows the distribution of feedback ranks (e.g., Excellent, Good, Poor), giving a visual summary of customer sentiment.

Map Integration: Users and employees can view parcel locations on Google Maps within the interface, enhancing spatial awareness and trust in delivery tracking.

Admin Interface (Expandable): Although not detailed in the current version, the system is designed to support administrative roles for managing users, overseeing courier operations, and analyzing performance metrics.

Technologies Used

Django: The web framework used to develop the server-side logic. It ensures secure, scalable, and maintainable code with MVC (Model-View-Controller) architecture.

Python: The programming language used for backend logic, data manipulation, and machine integration.

MySQL (via PyMySQL): The relational database used to store user data, courier details, feedback, and status logs.

Matplotlib: Used for generating pie charts to visualize feedback data.

NumPy: Employed to perform data aggregation and statistical calculations required for plotting graphs.

HTML/CSS (via Django Templates): Enables the frontend rendering of pages like feedback, parcel tracking, and status views.

Google Maps Embedding: Offers a geographical representation of parcel locations for easy visualization.

System Workflow

Parcel Collection: When a parcel is collected, an employee inputs all relevant data through a web form, including the image, and stores it in the database.

Status Updating: At each point of transit or delivery, the status is updated in the database along with the employee name, location, and date.

Tracking and Display: Customers or employees enter a parcel ID to retrieve detailed tracking information. Parcel details and current status are fetched and displayed in a structured table format, with options to view location on a map.

Feedback Mechanism: After delivery, customers can provide feedback through a simple form. Submitted data is used to create pie charts showing the overall quality of service.

Advantages

Real-time Tracking: Eliminates guesswork by keeping customers informed about their parcel's exact status.

Paperless Process: Moves away from manual records to a digitized system, reducing errors and saving time.

Improved Customer Satisfaction: Direct visibility, feedback options, and timely updates help build trust and improve service experience.

Actionable Analytics: Administrators can analyze feedback trends to enhance service quality.

Scalable and Secure: Built with Django, the system can easily be scaled to handle more users and parcels securely.

The Courier Management System presented here is a reliable, efficient, and user-friendly platform that addresses the key challenges in the courier industry. With end-to-end automation, map integration, and insightful feedback analytics, it provides a comprehensive toolset for courier companies to manage operations more effectively. The blend of modern web technologies and user-centered design ensures that the system can be a valuable asset in improving logistics operations and enhancing customer engagement Through this project, we demonstrate how Django and Python can be leveraged to build real-world applications that solve practical problems while offering scope for future scalability and innovation in the courier and logistics sector.

LITERATURE SURVEY

2.LITERATURE SURVEY

A **Courier Management System** is a digital solution that automates the processes involved in managing, tracking, and delivering parcels. Over the years, various courier tracking systems and logistics management platforms have been developed, each addressing specific aspects of the delivery pipeline. This literature survey explores existing systems, their technological frameworks, strengths, limitations, and how the proposed Django-based system builds upon

Traditional Courier Systems

Before digital transformation, courier services primarily relied on manual record-keeping and telephone-based status updates. Delivery slips and logbooks were used to track parcel movements. However, such systems had significant drawbacks:

Prone to human errors and data loss.

Delayed communication between the delivery team and customers.

No real-time tracking or feedback collection mechanism.

The inefficiencies in traditional courier management prompted the need for digital solutions that automate parcel tracking and status updates.

Web-Based Courier Systems

Several web-based courier tracking systems emerged to address the inefficiencies of manual systems. For instance:

Blue Dart and **DTDC** implemented customer-centric web platforms for basic tracking using consignment numbers.

FedEx and **DHL** introduced real-time global tracking features using barcodes and GPS.

While these platforms offered essential tracking functionality, they often had limitations:

Minimal user feedback mechanisms.

Limited visualization of tracking data.

Lack of transparency in location updates and service issues.

Restricted accessibility for internal employees to manage status updates through a unified portal.

Android-Based Tracking Applications

With the rise of smartphones, courier companies also introduced mobile apps that allowed users to:

Track shipments via GPS.

Get notifications on delivery status.

Access delivery history.

However, studies such as those by *Kumar et al. (2020)* showed that while mobile apps improved user engagement, they often failed to integrate seamlessly with internal systems used by courier staff, leading to data silos and inconsistencies.

Cloud-Based Logistics Platforms

Cloud-based systems such as **Zoho Logistics**, **ShipRocket**, and **ClickPost** provided advanced features:

Centralized databases with multi-user access.

AI-driven delivery optimization.

Integrated customer support ticketing.

Although highly efficient, these systems:

Are mostly subscription-based and costly for small businesses.

Require technical expertise for customization.

May not offer transparency into the backend for educational or open-source learning purposes.

Open Source Courier Systems

Open-source logistics platforms like **OpenBoxes** and **ParcelTrack** provide community-driven alternatives. They offer basic inventory and shipment tracking features but lack:

Integrated feedback visualization.

Image support for parcel identification.

Role-based access control with expandable modules for academic enhancement.

Academic and Research Projects

Numerous academic projects and prototypes have been developed, typically using technologies like:

PHP and MySQL for web development.

Android Studio for mobile-based tracking.

Firebase for real-time database handling.

Examples:

Sharma et al. (2021) designed a delivery tracking system using Firebase and GPS.

Rao and Singh (2019) proposed a logistics monitoring solution using IoT sensors and Java-based backends.

These projects demonstrated innovative ideas but were often either too narrow in scope (focusing only on tracking) or lacked user-friendly UIs and scalability.

Gap Analysis

From the existing literature and systems reviewed, several gaps are identified:

Limited real-time update capabilities for employees within many platforms.

Minimal or no integration of user feedback visualization using charts

Absence of parcel image verification features.

Lack of map-based tracking in some low-cost systems.

High deployment and maintenance costs in commercial solutions.

Little support for academic or prototype-level use in scalable frameworks like Django.

Proposed Solution Overview

To address the above gaps, this project presents a **Courier Management System using Django**, which includes:

- **Real-Time Status Updates**: Employees can update parcel location and delivery status from any web browser.
- **Parcel Image Upload**: Enhances verification and reduces parcel mix-ups.
- **Feedback Collection and Visualization**: Pie charts of customer feedback provide performance insights.
- **Google Maps Embedding**: Offers geographical visualization of parcel locations.
- **Secure, Role-Based Access**: Employees and users have distinct access and functionalities.
- **Scalable and Open-Source**: Suitable for academic use, small courier businesses, and further development.

While many existing systems focus either on the enterprise-scale logistics management or rudimentary tracking interfaces, there is a lack of fully integrated, open-source courier management systems with real-time interactivity, feedback analytics, and image/mapping support. The proposed Django-based Courier Management System bridges this gap by providing a modular, scalable, and user-friendly solution that combines real-world courier tracking needs with academic-level learning and customization.

SYSTEM ANALYSIS AND DESIGN

3.SYSTEM ANALYSIS AND DESIGN

System analysis is a critical phase in the software development life cycle (SDLC) that involves understanding and defining the problems, requirements, and operational environment of a proposed system. This phase sets the foundation for designing and developing an effective solution. In this context, the courier management system is analyzed from functional, non-functional, and feasibility standpoints to ensure the system meets stakeholder expectations.

3.1 EXISTING SYSTEM

The current courier management processes in many organizations are often semi-automated or entirely manual. In such systems, courier booking, tracking, and delivery updates are handled through physical paperwork, phone calls, or basic spreadsheet tools. This traditional approach has several limitations, especially in handling a growing number of shipments and maintaining customer satisfaction.

Problem Definition

In the current courier industry, especially among small and medium-sized enterprises, there are several pain points:

Inefficient manual handling of shipments.

Lack of real-time tracking for parcels.

Minimal customer interaction or feedback mechanisms.

Absence of parcel image verification features.

No geographic visualization (map integration).

High cost and complexity of proprietary software solutions.

These issues lead to reduced customer satisfaction, operational inefficiency, and lack of accountability.

3.2 PROPOSED SYSTEM

The proposed system is a web-based **Courier Management System** built using **Django (Python)** and **MySQL**, aimed at automating and streamlining the entire parcel management lifecycle—from booking and dispatch to tracking and feedback collection. The proposed system allows customers to book parcels online, generate tracking IDs instantly, and monitor the real-time status of their shipments. It enables staff members to update parcel status, assign delivery personnel, and manage daily tasks through an intuitive dashboard. Administrators have access to system-wide reports, user management tools, and complete control over courier workflows.

KEY FEATURES

Online Parcel Booking and Tracking.

Real-Time Status Updates and Notifications.

Role-Based Access for Admin, Staff, and Customers.

Automated Reports and Delivery Logs.

Secure Login and Data Management.

Parcel image uploads for verification.

Feedback collection with graphical analysis (pie charts).

Location visualization using embedded Google Maps.

Objectives of the System

Automate courier tracking and delivery process.

Provide customers with live updates and visual data.

Enable staff to upload parcel images and update status.

Display feedback trends to help improve services.

Offer an easy-to-use, secure, and scalable system.

Feasibility Study

Technical Feasibility

Frontend: HTML, CSS, JavaScript

Backend: Django (Python Framework)

Database: MySQL

Tools: VS Code, XAMPP (MySQL), Python 3.x, Git

All technologies are open-source and supported by large communities, ensuring maintainability and extensibility.

Operational Feasibility

The system will streamline courier operations and reduce human errors.

Easy to use interface for all user roles ensures smooth adoption.

The feedback and tracking features improve customer engagement.

Economic Feasibility

As an open-source and academic project, the system eliminates software licensing costs.

Can be hosted on low-cost servers or local systems for small-scale deployment.

Functional Requirements

Admin:

Manage employee and customer accounts.

View and delete parcels and feedback.

View system-wide analytics and performance.

Employee:

Add new courier details (parcel name, receiver, sender).

Upload parcel images.

Update status and location of parcels.

Customer:

Register and log in to view personal courier records.

Track parcel location and status in real-time.

Submit feedback on courier services.

Non-Functional Requirements

Usability: Clean and intuitive UI for all users.

Performance: Quick data fetch and update response time.

Security: Role-based authentication, input validation, and secure storage of user credentials.

Scalability: Designed to support multiple users simultaneously and handle increased parcel records.

Maintainability: Modular Django apps make future updates and debugging easier.

SWOT Analysis

Strengths		**Weakness**
Easy to use	:	Limited offline capabilities
Real-time status tracking	:	Lacks mobile app(web-only)
Open-source and low cost	:	Internet-dependent for updates

Opportunities	**Threats**
Can be scaled to multiple branches	Security risks if not regularly updated
Can integrate barcode/RFID in future	Server downtime could impact real-time features

Context Flow Diagram (Level 0)

[Customer] → (Login/Track Parcel/Feedback) → **Courier Management System** ← (Add/Update Parcel, Upload Image) ← **Employee**

Admin ← (Manage Users, View Feedback, Reports) ← **Courier Management System**

Entity-Relationship Overview (high-level)

User (User_ID, Name, Role, Email, Password)

Courier (Courier_ID, Sender, Receiver, Location, Status, Employee_ID)

Image (Image_ID, Courier_ID, File_Path, Upload_Date)

Feedback (Feedback_ID, User_ID, Rating, Comment)

The system analysis reveals that a Django-based courier management system is both feasible and highly beneficial. It improves operational efficiency, enhances customer experience through real-time updates and feedback mechanisms, and provides a scalable solution for courier companies and academic use. The clear separation of roles and functions ensures the system meets diverse user needs while being secure and extensible for future enhancements.

3.3 Feasibility Study

The feasibility of the project is analyzed in this phase and business proposal is put forth with a very general plan for the project and some cost estimates. During system analysis the feasibility study of the proposed system is to be carried out. This is to ensure that the proposed system is not a burden to the company. For feasibility analysis, some understanding of the major requirements for the system is essential.

Three key considerations involved in the feasibility analysis are,

- **ECONOMICAL FEASIBILITY**
- **TECHNICAL FEASIBILITY**
- **SOCIAL FEASIBILITY**

3.3.1 Economic Feasibility

This study is carried out to check the economic impact that the system will have on the organization. The amount of fund that the company can pour into the research and development of the system is limited. The expenditures must be justified. Thus the developed system as well within the budget and this was achieved because most of the technologies used are freely available. Only the customized products had to be purchased.

3.3.2 Technical Feasibility

This study is carried out to check the technical feasibility, that is, the technical requirements of the system. Any system developed must not have a high demand on the available technical resources. This will lead to high demands on the available technical resources. This will lead to high demands being placed on the client. The developed system must have a modest requirement, as only minimal or null changes are required for implementing this system.

3.3.3 Social Feasibility

The aspect of study is to check the level of acceptance of the system by the user. This includes the process of training the user to use the system efficiently. The user must not feel threatened by the system, instead must accept it as a necessity. The level of acceptance by the users solely depends on the methods that are employed to educate the user about the system and to make him familiar with it. His level of confidence must be raised so that he is also able to make some constructive criticism, which is welcomed, as he is the final user of the system.

SYSTEM DESIGN

System design is the blueprint of a software system that bridges the gap between system requirements and actual implementation. It lays out the architecture, modules, data flow, and interaction of components to ensure the software functions efficiently, securely, and as expected. This Courier Management System includes both **High-Level Design (HLD)** and **Low-Level Design (LLD)** components to detail the application's architecture and internal structure.

Architectural Design

The system follows the **Three-Tier Architecture**:

Presentation Layer: User Interface (HTML/CSS/JS) – interacts with users.

Application Layer: Django Views and Business Logic – handles user input, control application flow.

Data Layer: MySQL – stores user, courier, image, and feedback data.

Module-wise Design

Admin Module

View and manage employees and customers.

Access all courier records and feedback.

View analytics dashboard (e.g., pie charts of feedback).

Delete or update courier data or users.

Employee Module

Add new courier records (sender, receiver, parcel description).

Upload images of parcels.

Update courier status and location in real-time

Customer Module

Register/Login to track parcels.

View current courier status and image.

Submit service feedback (ratings and comments).

Use Case Diagram

```
          +---------------------+
          |     <<Actor>>       |
          |      Customer       |
          +----------+----------+
                     |
        +-------------+-------------+
        |                           |
+--------v--------+      +-----------v-----------+
|  Track Courier  |      |    Submit Feedback    |
+-----------------+      +----------------------+


          +-------------------+
          |    <<Actor>>      |
          |    Employee       |
          +----------+----------+
                     |
      +--------------+---------------+
      |                              |
+-------v-------+           +-----------v-----------+
|  Add Courier  |           |   Upload Parcel Img   |
+---------------+           +-----------------------+
                            | Update Courier Status  |
                            +-----------------------+


          +-------------------+
          |    <<Actor>>      |
          |      Admin        |
          +----------+----------+
                     |
      +--------------+--------------+
      |              |              |
+-------v------+ +------v--------+ +----v------+
| Manage Users | | View Feedback | | Analytics |
+--------------+ +---------------+ +-----------+
```

Class Diagram (Simplified)

```
+-----------------+
|      User       |
+-----------------+
| user_id         |
| name            |
| email           |
| password        |
| role            |
+-----------------+


+-----------------+
|     Courier     |
+-----------------+
| courier_id      |
| sender          |
| receiver         |
| location         |
| status           |
| employee_id      |
+------------------+

+------------------+
|     Image        |
+------------------+
| image_id         |
| courier_id       |
| file_path        |
| upload_date      |
+------------------+

+------------------+
|    Feedback      |
+------------------+
| feedback_id      |
| user_id          |
| rating           |
| comment          |
+------------------+
```

Database Design (Schema)

User Table

user_id (Primary Key)

name

email

password

role (Admin, Employee, Customer)

Courier Table

courier_id (Primary Key)

sender

receiver

location

status

employee_id (Foreign Key)

Image Table

image_id (Primary Key)

courier_id (Foreign Key)

file_path

upload_date

Feedback Table

feedback_id (Primary Key)

user_id (Foreign Key)

rating (1–5)

comment

- ## Data Flow Diagram (Level 1)

```
[Customer] ---> (Login/Register) ---> [Django System] ---> [User DB]
        ---> (Track Parcel) ---> [Django System] ---> [Courier DB]
        ---> (Submit Feedback) ---> [Feedback DB]

[Employee] ---> (Add Courier) ---> [Courier DB]
        ---> (Upload Image) ---> [Image DB]
        ---> (Update Status) ---> [Courier DB]

[Admin] ---> (Manage Users/Couriers) ---> [User/Courier DB]
      ---> (View Feedback) ---> [Feedback DB]
      ---> (Analytics) ---> [Feedback/Stats]
```

Sequence Diagram (Add Courier Example)

```
Customer --> Employee: Request Courier Booking
Employee --> System: Login
Employee --> System: Add Courier (Details)
System --> Database: Insert into Courier Table
Database --> System: Confirmation
System --> Employee: Success Message
```

Frontend Page Design (Basic UI Flow)

Admin

Dashboard (total users, parcels, feedback stats)

Manage Users (Add/Delete/Edit)

View Courier List

View Feedback Pie Chart

Employee

Add Courier Form

Upload Parcel Image

Update Status and Location

Customer

Register/Login Page

My Couriers List

Courier Details with Image

Feedback Form

Security Considerations

Role-based access control using Django's authentication system.

CSRF protection on all forms.

Password hashing using Django’s built-in security features.

Input validation and form sanitization.

The system design of the Courier Management System ensures a modular, scalable, and secure architecture. Each component is clearly defined to handle specific tasks efficiently while interacting seamlessly with others. The use of Django's MVC framework and MySQL as a robust relational database supports rapid development and long-term maintainability.

3.4 SYSTEM REQUIREMENTS

3.4.1 Functional Requirements

Admin can manage user roles, monitor courier entries, and view reports.

Employees can add, update, and track courier records and upload delivery proofs.

Customers can track shipments and submit feedback.

Role-based authentication for users (admin, employee, customer).

Image upload and feedback submission functionality.

Auto-update of courier status.

3.4.2 Non-Functional Requirements

Performance: Quick response to CRUD operations, even with 1000+ entries.

Scalability: Supports scaling up for multi-branch use.

Security: User login authentication and permission checks.

Usability: Simple UI for non-technical staff and customers.

Reliability: Handles multiple user sessions and real-time status updates.

3.4.3 Hardware Requirements

Processor: Intel i5 or higher

RAM: 8 GB or more

Storage: 100 GB HDD or SSD

Network: Stable internet for deployment and access

3.4.4 Software Requirements

OS: Windows/Linux/Mac

Python 3.x

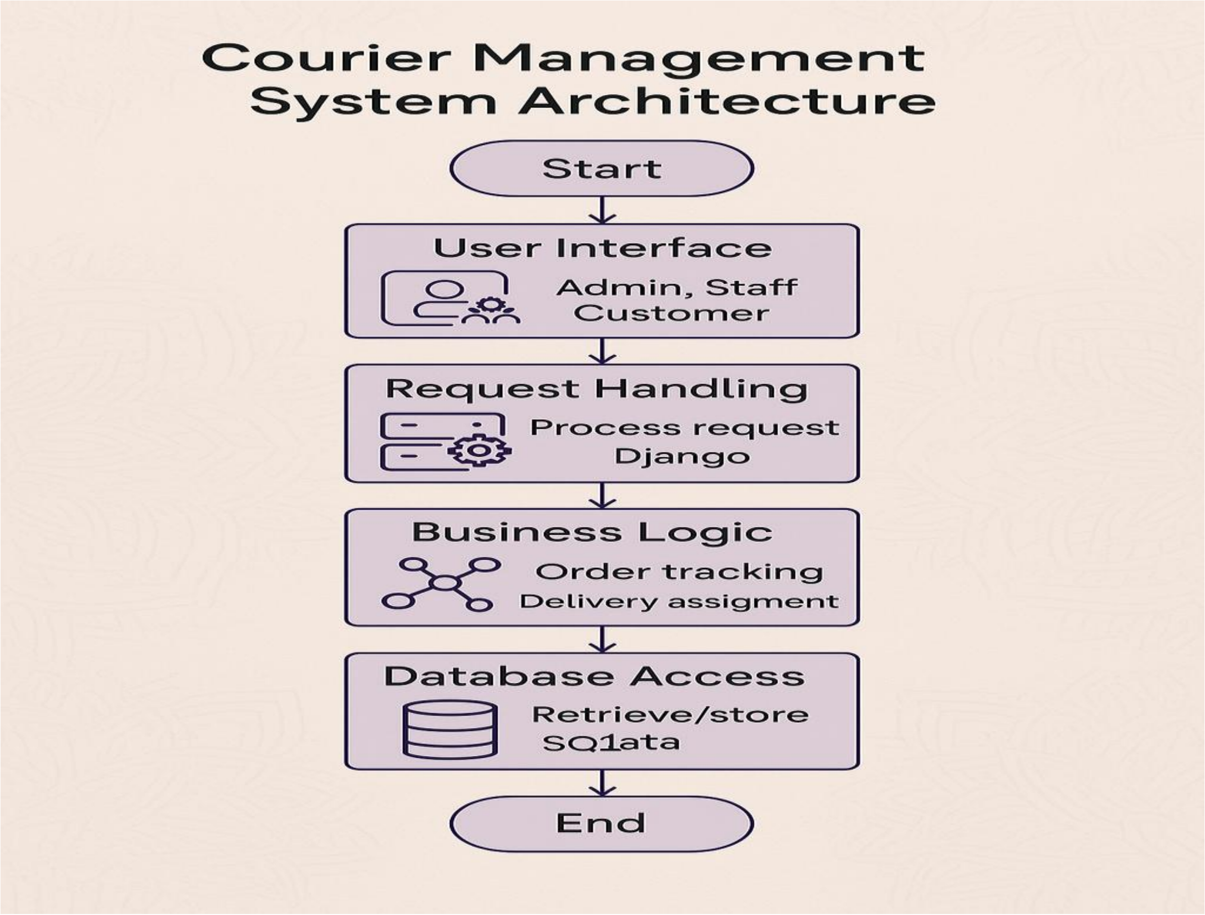
Django 3.0+

MySQL or MariaDB

Web Browser (Chrome, Firefox)

Git, VS Code or PyCharm

3.5 SYSTEM ARCHITECTURE



This is a System Architecture Diagram for a Courier Management System, and it illustrates the flow of data and control through the system's layers—from start to finish. It's structured as a linear pipeline from user input to data storage.

The diagram consists of 5 main stages:

1. Start

Represents the beginning of the process when a user initiates interaction with the system (e.g., logging in, tracking a courier, etc.).

2.User Interface

Actors: Admin, Staff, Customer

Purpose: Front-end interface where users interact with the system

Admins might manage orders and users

Staff might handle delivery assignments

Customers might track their couriers

Typically implemented using web frameworks (HTML/CSS/JS or frontend libraries).

3. Request Handling

Function: Handles incoming user requests and routes them appropriately

Technology: Django

Receives requests from the UI (e.g., form submissions, clicks)

Processes them through Django views and URLs

This is the Controller in MVC (Model-View-Controller) architecture.

4. Business Logic

Functions:

Order tracking: Track status of deliveries

Delivery assignment: Assign couriers or staff to specific deliveries

Core functionality layer, where the main rules and operations of the system are enforced

Implemented in Django models or service functions

5. Database Access

Function: Retrieve and store data

Database Used: SQLite (spelled as “SQ1ata” by mistake; should be SQLite)

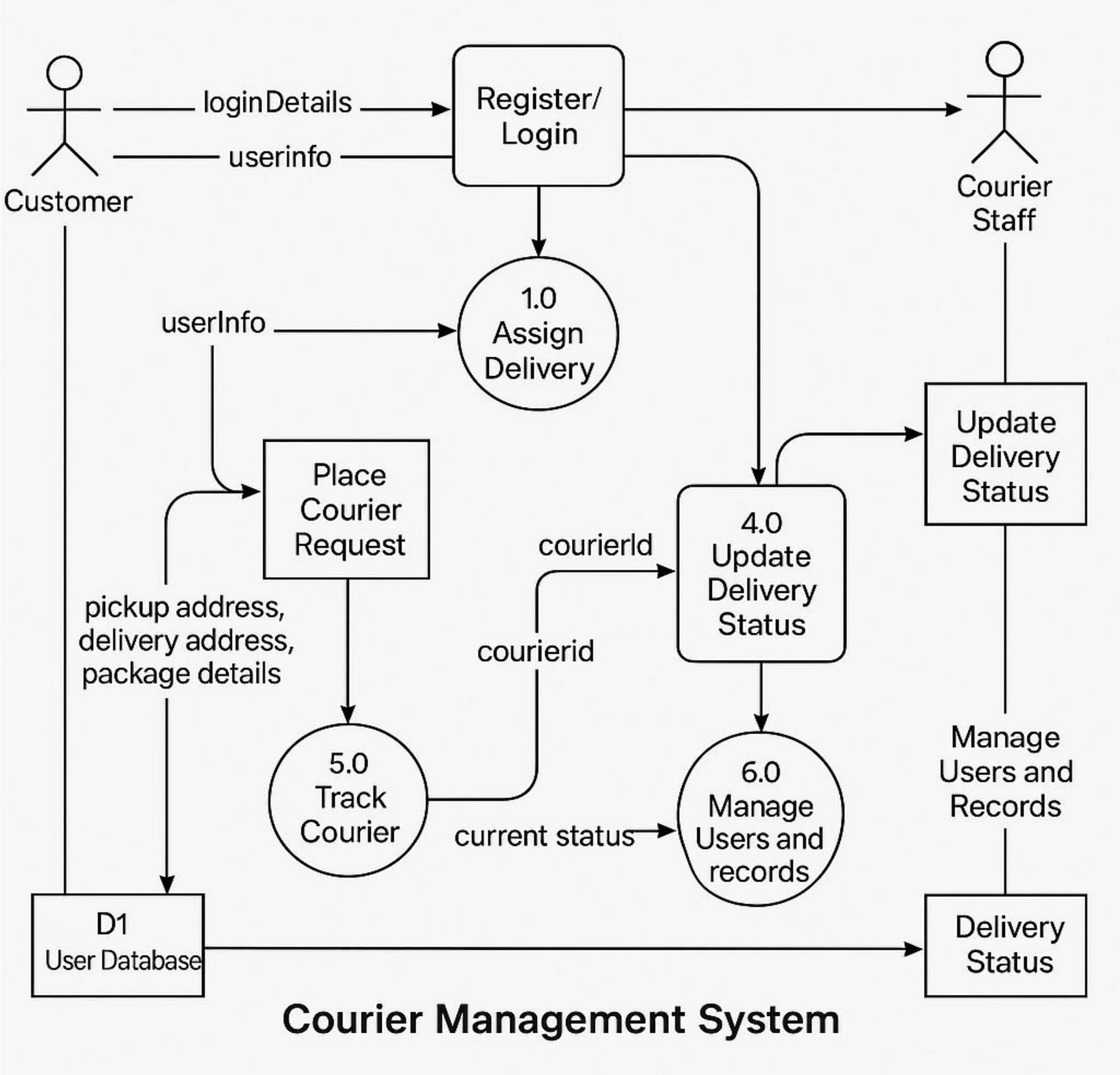
Handles CRUD operations (Create, Read, Update, Delete)

Used by Django ORM to interact with the database

6. End

Represents completion of the process, like showing the user the result (e.g., delivery status, successful assignment, etc.)

3.5 DATAFLOW DIAGRAM



Courier Management System - Level 1 DFD

External Entities:

Customer

Admin

Courier Staff

Processes:

1.0 Register/Login

Users (Customer/Admin/Staff) login or register.

Data: loginDetails, userInfo

2.0 Place Courier Request

Customer places a delivery request.

Data: pickup address, delivery address, package details

3.0 Assign Delivery

Admin assigns the delivery to courier staff.

Data: courierId, assigned package

4.0 Update Delivery Status

Staff updates the delivery status.

Data: in transit, delivered, delayed, etc.

3.7 SOFTWARE DEVELOPMENT LIFECYCLE(SDLC)

The Software Development Life Cycle (SDLC) is a systematic and structured framework followed by software engineers, project managers, and organizations to design, develop, test, and deploy high-quality software. It provides a step-by-step process to ensure that software meets both technical and user requirements, functions efficiently, and can be maintained and improved over time.

SDLC helps manage complexity, reduce risks, control budgets, and streamline team collaboration. It is essential for delivering successful software systems — from small web applications to large-scale enterprise solutions like a Courier Management System.

Functional Requirements

Graphical User interface with the User.

Non-Functional Requirements

- **Maintainability:** Maintainability is used to make future maintenance easier, meet new requirements. Our project can support expansion.
- **Robustness:** Robustness is the quality of being able to withstand stress, pressures or changes in procedure or circumstance. Our project also provides it.
- **Reliability:** Reliability is an ability of a person or system to perform and maintain its functions in circumstances. Our project also provides it.

- **Size:** The size of a particular application plays a major role, if the size is less then efficiency will be high. The size of database we have developed is 5.05 MB.
- **Speed:** If the speed is high then it is good. Since the no of lines in our code is less, hence the speed is high.
- **Power Consumption:** In battery-powered systems, power consumption is very important. In the requirement stage, power can be specified in terms of battery life.

However the allowable wattage can’t be defined by the customer. Since the no of lines of code is less CPU uses less time to execute hence power usage will be less.

3.8 UML DIAGRAMS

UML stands for Unified Modeling Language. UML is a standardized general-purpose modeling language in the field of object-oriented software engineering. The standard is managed, and was created by, the Object Management Group.

The goal is for UML to become a common language for creating models of object oriented computer software. In its current form UML is comprised of two major components: a Meta-model and a notation. In the future, some form of method or process may also be added to; or associated with, UML.

The Unified Modeling Language is a standard language for specifying, Visualization, Constructing and documenting the artifacts of software system, as well as for business modeling and other non-software systems.

The UML represents a collection of best engineering practices that have proven successful in the modeling of large and complex systems.

The UML is a very important part of developing objects oriented software and the software development process. The UML uses mostly graphical notations to express the design of software projects.

GOALS:

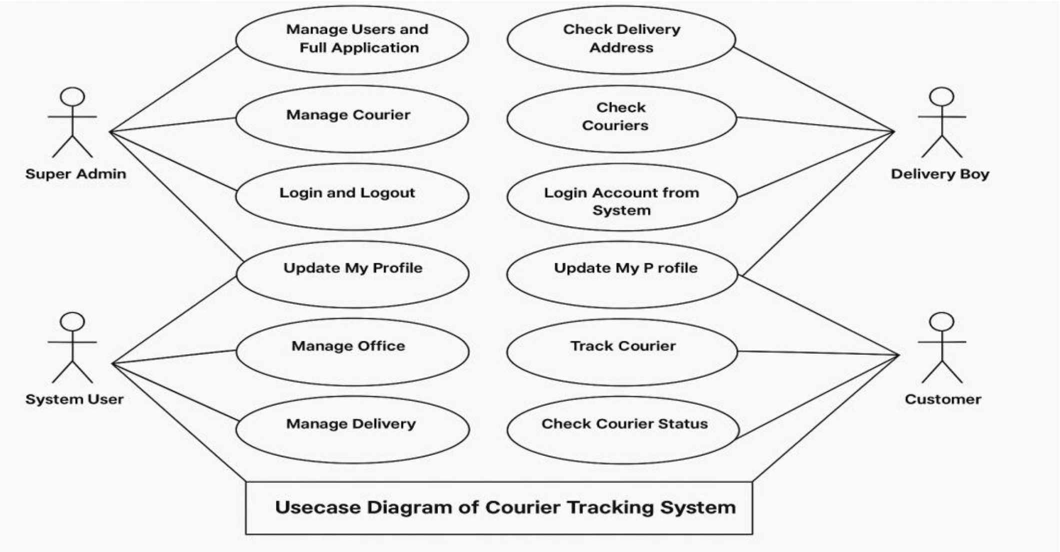
The Primary goals in the design of the UML are as follows:

1. Provide users a ready-to-use, expressive visual modeling Language so that they can develop and exchange meaningful models.
2. Provide extendibility and specialization mechanisms to extend the core concepts.
3. Be independent of particular programming languages and development process.
4. Provide a formal basis for understanding the modeling language.
5. Encourage the growth of OO tools market.

6.Support higher level development concepts such as collaborations, frameworks, patterns and components.

7. Integrate best practices.

3.8.1 USE CASE DIAGRAM



This is a Use Case Diagram for a Courier Tracking System. It visually represents the different users (actors) of the system and the interactions (use cases) they have with the system.

Actors (Users of the System)

Super Admin

Full control over the application and user management.

System User

Manages internal operations like office and delivery.

Delivery Boy

Interacts with couriers and delivery info.

Customer

Tracks courier and checks delivery status.

Use Cases (System Features)

For Super Admin:

Manage Users and Full Application

Handles user roles, permissions, and system-wide settings.

Manage Courier

Add, update, or remove courier records.

Login and Logout

Basic authentication functionality.

Update My Profile

Change account information.

For System User:

Login and Logout

Sign in/out of the system.

Update My Profile

Modify personal details.

Manage Office

Control office-related data, locations, and branches.

Manage Delivery

Assign, monitor, and update delivery tasks.

For Delivery Boy:

Check Delivery Address. Get delivery location details.

Check Couriers

View courier assignments.

Login Account from System

Sign into the delivery interface.

Update My Profile

Change contact or personal info.

For Customer:

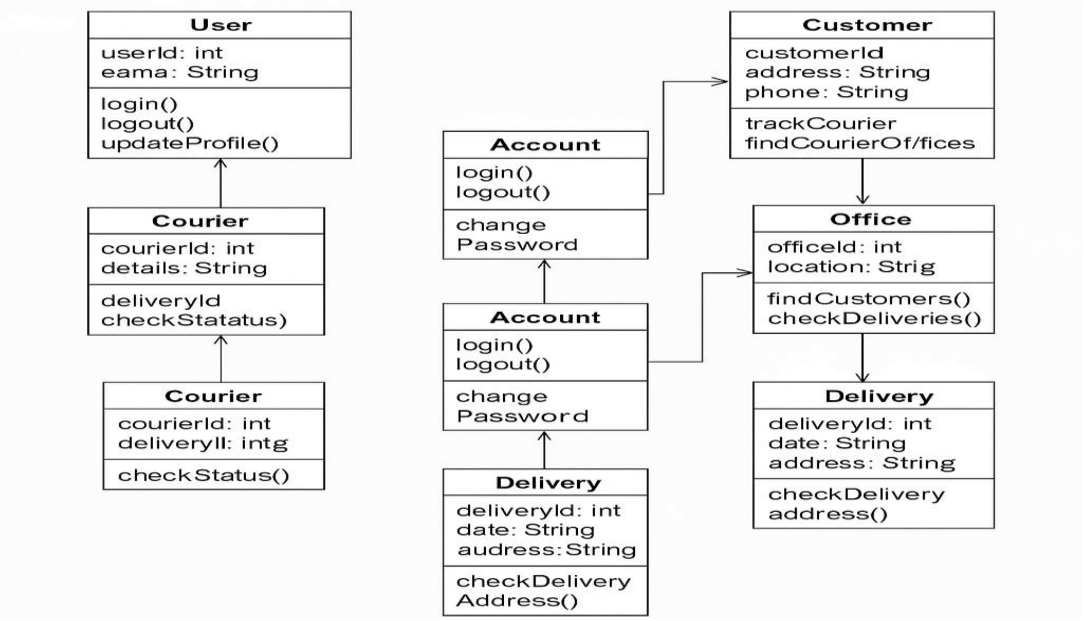
Track Courier

Monitor real-time courier location or status.

Check Courier Status

View if courier is dispatched, in transit, or delivered.

3.8.2 CLASS DIAGRAM



This is a Class Diagram for a Courier Tracking System, and it visually represents the system's key classes, their attributes, methods (functions), and relationships between them. Let's break it down:

Class Components Overview

Each box represents a class, containing:

Attributes (variables) at the top

Methods (functions) at the bottom

Main Classes & Their Roles

1. **User Attributes:** userId, email (typo: "eama" should be "email")

Methods:

login(), logout(), updateProfile()

Represents general users of the system, possibly admins or staff.

2. Courier

Attributes:

courierId, details, deliveryId

Methods:

checkStatus()

Linked with Delivery to track parcel status.

3. Customer

Attributes:

customerId, address, phone

Methods:

trackCourier(), findCourierOffices()

Represents end-users tracking and managing deliveries.

4. Account

Methods only: login(), logout(), changePassword()

Represents authentication features for users, customers, etc.

Appears multiple times, showing its reusability across other entities.

5. Delivery

Attributes:

Delivery Id, date, address (typo: “address” should be “address”)

Methods:

checkDeliveryAddress()

Represents the parcel or item being delivered.

Also appears twice, likely due to diagram design layout.

6. Office

Attributes:

officeId, location (typo: “Strig” should be “String”)

Methods:

Find Customers(), check Deliveries()

Represents courier branches handling deliveries and customer queries.

Relationships

User → Manages Courier

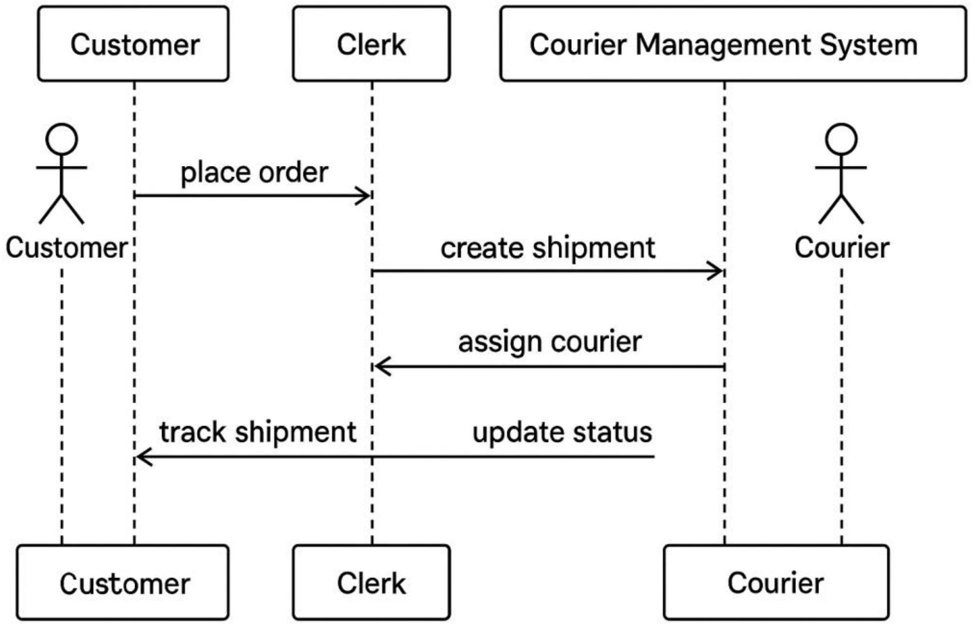
Courier → Has a Delivery Id, linked to Delivery

Customer → Can interact with Office and Delivery

Office → Handles Delivery and Customer

Account → Associated with all major user types (User, Customer, Courier) for login/authentication.

3.8.3 SEQUENCE DIAGRAM



A sequence diagram for a Courier Management System would typically illustrate interactions between various entities such as the customer, courier staff, admin, and the system. Here’s an overview of how you might structure the sequence diagram for common operations in such a system:

Key Participants:

Customer

Courier Staff

Admin

System

Database

Main Actions:

Parcel Collection – Customer initiates the collection request.

Courier Assignment – Admin assigns a courier to pick up the parcel.

Courier Pickup – Courier picks up the parcel.

Parcel Delivery – Courier delivers the parcel to the recipient.

Feedback – Customer provides feedback.

Sequence of Events:

Customer Initiates Collection:

Customer sends a "Request for Collection" to the system.

System stores the request and updates the database.

Admin Assigns Courier:

Admin logs into the system and assigns the courier to the request.

System updates the status and sends a notification to the assigned courier.

Courier Picks Up Parcel:

Courier receives the notification and confirms pickup.

Courier updates the system with parcel pickup status.

Parcel Delivery:

Courier delivers the parcel to the recipient.

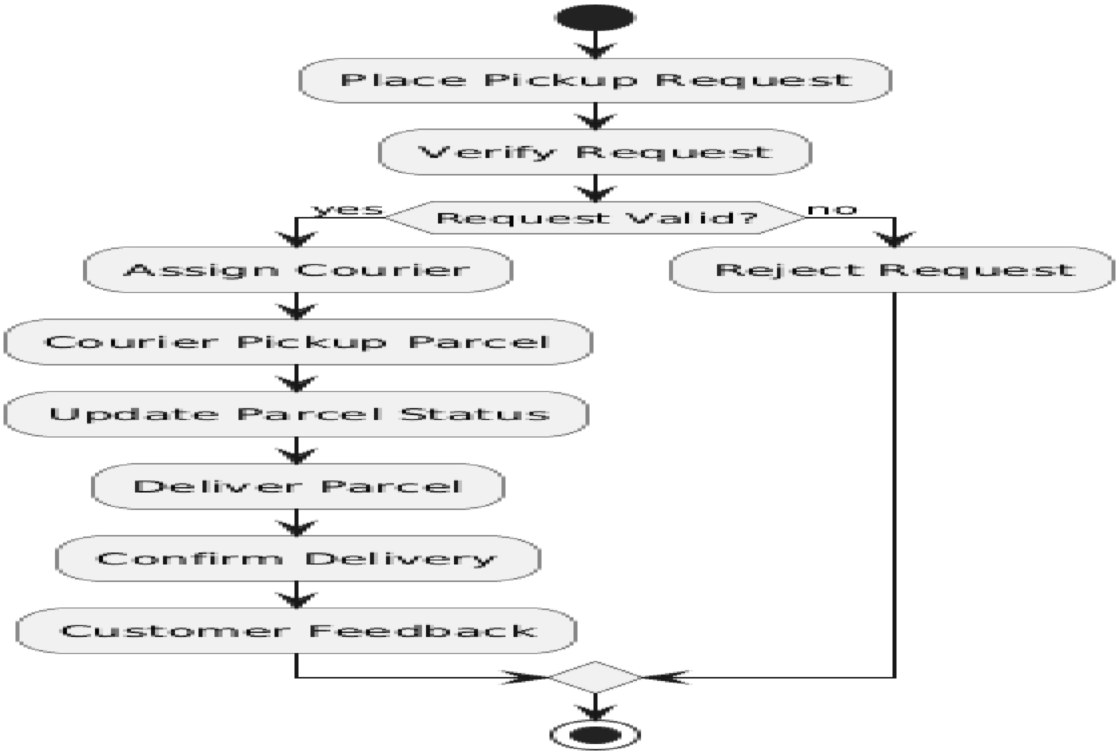
Courier updates delivery status in the system.

Feedback:

Customer provides feedback about the delivery.

System stores feedback in the database.

3.8.4 ACTIVITY DIAGRAM



Start Node: Denotes the beginning of the process when a customer initiates a courier service request.

Place Pickup Request: The customer fills in details like pickup and drop-off locations, parcel type, weight, etc.

Verify Request: The system checks for completeness and validity of the request (e.g., service area coverage, weight limits).

Decision Node: If the request is valid, it proceeds; otherwise, it's rejected.

Assign Courier: Admin (or system) assigns a courier based on availability and proximity.

Pickup Parcel: The courier collects the parcel from the sender.

Update Parcel Status: The system updates the tracking status (e.g., "Picked Up").

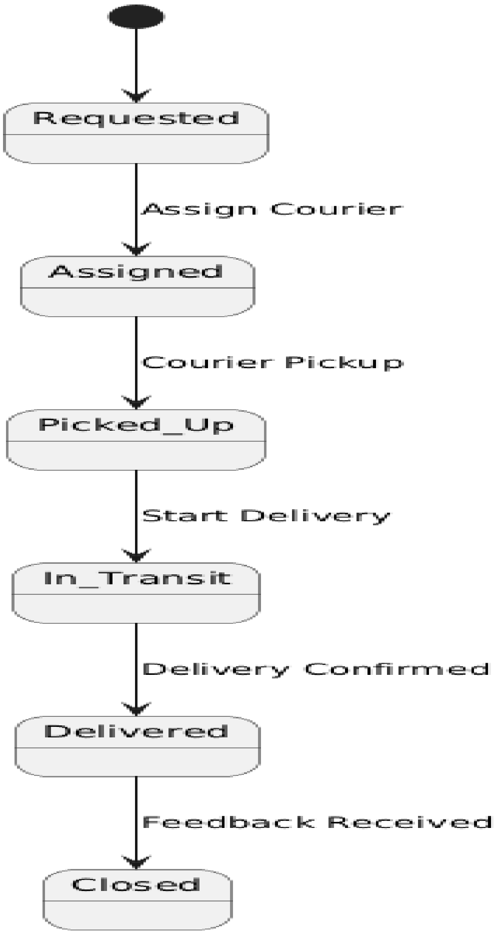
Deliver Parcel: The courier transports and delivers the parcel to the recipient.

Confirm Delivery: Delivery is confirmed either by signature or OTP verification.

Customer Feedback: After delivery, the customer can rate and review the service.

Stop Node: Ends the workflow after delivery and feedback.

3.8.5 STATECHART DIAGRAM



Requested: Initial state after a customer submits a courier request.

Assigned: An admin (or system) assigns a courier to the parcel.

Picked Up: Courier has collected the parcel from the customer.

In Transit: The parcel is on the way to the destination.

Delivered: The courier successfully hands over the parcel.

Closed: Feedback is received, and the transaction is finalized

Each transition is triggered by an event:

“Assign Courier” moves the parcel from Requested to Assigned.

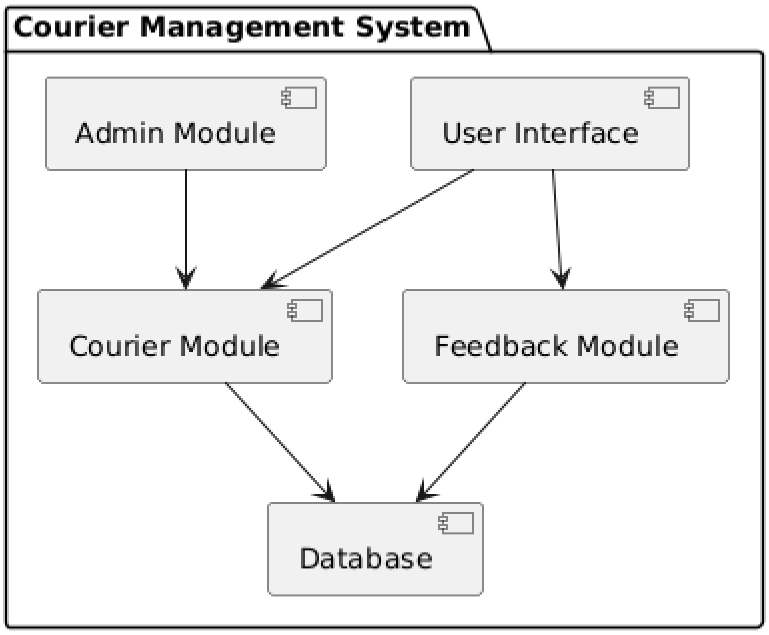
“Courier Pickup” triggers the move to Picked Up.

"Start Delivery" begins the In Transit state.

"Delivery Confirmed" moves it to Delivered.

"Feedback Received" ends the process in the Closed state.

3.8.6 COMPONENT DIAGRAM



User Interface (UI): The frontend accessed by customers, admins, and couriers. It sends requests and displays responses.

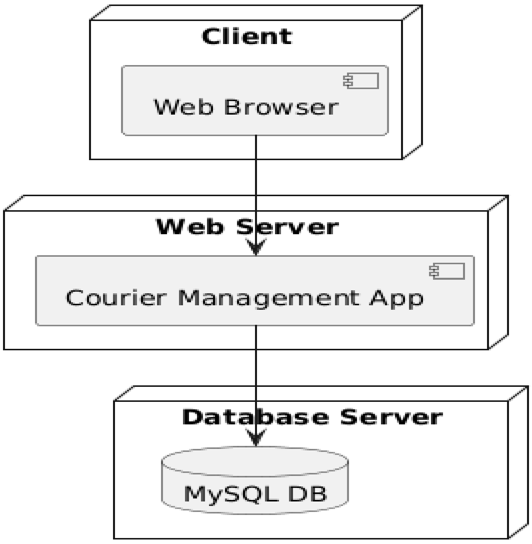
Courier Module: Handles parcel creation, tracking, assignment, and updates.

Admin Module: Lets the admin manage couriers, view requests, and assign tasks.

Feedback Module: Manages customer reviews and satisfaction surveys.

Database: Stores user data, courier details, parcel logs, feedback, etc.

3.8.7 DEPLOYEMENT DIAGRAM



Client Node: The customer's or admin's device (laptop, mobile). Runs a web browser that accesses the web app.

Web Server Node: Hosts the main Courier Management Application. Handles HTTP requests, authentication, and business logic.

Database Server Node: Stores all persistent data such as parcel status, user info, and transaction history. Usually a separate server for scalability and security.

The diagram shows how these parts are connected and communicate (e.g., REST API calls between client and server, SQL queries between app and DB).

3.9 INPUT DESIGN

The input design is the link between the information system and the user. It comprises the developing specification and procedures for data preparation and those steps are necessary to put transaction data in to a usable form for processing can be achieved by inspecting the computer to read data from a written or printed document or it can occur by having people keying the data directly into the system. The design of input focuses on controlling the amount of input required, controlling the errors, avoiding delay, avoiding extra steps and keeping the process simple. The input is designed in such a way so that it provides security and ease of use with retaining the privacy. Input Design considered the following things:

- What data should be given as input?
- How the data should be arranged or coded?
- The dialog to guide the operating personnel in providing input.
- Methods for preparing input validations and steps to follow when error occur.

OBJECTIVES

Input Design is the process of converting a user-oriented description of the input into a computer-based system. This design is important to avoid errors in the data input process and show the correct direction to the management for getting correct information from the computerized system.

It is achieved by creating user-friendly screens for the data entry to handle large volume of data. The goal of designing input is to make data entry easier and to be free from errors. The data entry screen is designed in such a way that all the data manipulates can be performed. It also provides record viewing facilities.

When the data is entered it will check for its validity. Data can be entered with the help of screens. Appropriate messages are provided as when needed so that the user will not be in maize of instant. Thus the objective of input design is to create an input layout that is easy to follow.

3.10 OUTPUT DESIGN

A quality output is one, which meets the requirements of the end user and presents the information clearly. In any system results of processing are communicated to the users and to other system through outputs. In output design it is determined how the information is to be displaced for immediate need and also the hard copy output. It is the most important and direct source information to the user. Efficient and intelligent output design improves the system’s relationship to help user decision-making.

Designing computer output should proceed in an organized, well thought out manner; the right output must be developed while ensuring that each output element is designed so that people will find the system can use easily and effectively. When analysis design computer output, they should Identify the specific output that is needed to meet the requirements.

Select methods for presenting information.

Create document, report, or other formats that contain information produced by the system.

The output form of an information system should accomplish one or more of the following objectives.

- Convey information about past activities, current status or projections of the
- Future.
- Signal important events, opportunities, problems, or warnings.
- Trigger an action.
- Confirm an action.

IMPLMENTATION

4.IMPLEMENTATION

4.1 MODULES

1. User Authentication Module:

Handles login/logout for all users (Admin, Customer, Delivery Boy, Manager)

Role-based access control

2. Courier Management Module:

Book new courier entries

Generate tracking numbers

Update courier status (Booked, In Transit, Delivered, etc.)

Assign couriers to delivery staff

3. Customer Management Module:

Add/edit customer details

View courier history for customers

Search couriers by customer ID or name

4. Tracking Module:

Public or customer portal to enter tracking ID

Show current status and courier history

Estimated delivery date

5. Delivery Management Module:

Assign couriers to delivery boys

Update delivery status

Track delivery locations (optional GPS integration)

6. Office/Branch Management Module:

Manage branch locations and contact info

Link couriers to specific branches

Assign managers to offices

7. User Management Module (Admin only):

Add, edit, or delete system users

Assign roles: Admin, Manager, Delivery Boy

8. Shipment Module:

Handle detailed shipment info: weight, price, type (fragile, express)

Calculate charges

Print invoice or receipt

9. Reports Module:

Generate reports:

Daily courier list

Delivery success/failure rate

Staff performance

Export to PDF or Excel

10. Notification Module:

Email/SMS alerts for status updates

System Design Implementation

The implementation of the Courier Management System involves translating the system design into actual, functional code using Django (a Python-based web framework) and MySQL (a relational database). The system is modular and uses the MVC (Model-View-Controller) architecture pattern.

Technology Stack

Frontend: HTML, CSS, JavaScript, Bootstrap

Backend: Python (Django Framework)

Database: MySQL

Tools: Visual Studio Code, XAMPP (for MySQL), Git

Django App Structure

```
CourierManagement/
├── courier_app/
│   ├── migrations/
│   ├── static/
│   ├── templates/
│   │   ├── admin/
│   │   ├── employee/
│   │   ├── customer/
│   ├── __init__.py
│   ├── admin.py
│   ├── apps.py
│   ├── models.py
│   ├── views.py
│   ├── forms.py
│   ├── urls.py
```

```
├── CourierManagement/
│   ├── __init__.py
│   ├── settings.py
│   ├── urls.py
│   ├── wsgi.py
│   ├── asgi.py
├── manage.py
```

Models (models.py)

```
from django.db import models
from django.contrib.auth.models import User
class Courier(models.Model):
    sender = models.CharField(max_length=100)
    receiver = models.CharField(max_length=100)
    location = models.CharField(max_length=100)
    status = models.CharField(max_length=50)
    assigned_to = models.ForeignKey(User, on_delete=models.CASCADE)
class ImageUpload(models.Model):
    courier = models.ForeignKey(Courier, on_delete=models.CASCADE)
    image = models.ImageField(upload_to='uploads/')
    uploaded_at = models.DateTimeField(auto_now_add=True)
class Feedback(models.Model):
    user = models.ForeignKey(User, on_delete=models.CASCADE)
    rating = models.IntegerField()
    comment = models.TextField()
    date = models.DateTimeField(auto_now_add=True)
```

Forms (forms.py)

```
from django import forms
from .models import Courier, Feedback, ImageUpload
class CourierForm(forms.ModelForm):
    class Meta:
        model = Courier
        fields = ['sender', 'receiver', 'location', 'status']
class ImageForm(forms.ModelForm):
    class Meta:
        model = ImageUpload
        fields = ['courier', 'image']
class FeedbackForm(forms.ModelForm):
    class Meta:
        model = Feedback
        fields = ['rating', 'comment']
```

Views (views.py)

```
from django.shortcuts import render, redirect
from .models import Courier, Feedback, ImageUpload
from .forms import CourierForm, FeedbackForm, ImageForm
from django.contrib.auth.decorators import login_required
@login_required
def dashboard(request):
    if request.user.is_superuser:
        return render(request, 'admin/dashboard.html')
    elif request.user.groups.filter(name='Employee').exists():
        return render(request, 'employee/dashboard.html')
    else:
        return render(request, 'customer/dashboard.html')
@login_required
def add_courier(request):
    if request.method == 'POST':
        form = CourierForm(request.POST)
        if form.is_valid():
            courier = form.save(commit=False)
            courier.assigned_to = request.user
            courier.save()
            return redirect('dashboard')
    else:
        form = CourierForm()
    return render(request, 'employee/add_courier.html', {'form': form})
@login_required
def upload_image(request):
    if request.method == 'POST':
        form = ImageForm(request.POST, request.FILES)
        if form.is_valid():
            form.save()
            return redirect('dashboard')
    else:
        form = ImageForm()
    return render(request, 'employee/upload_image.html', {'form': form})
@login_required
def submit_feedback(request):
    if request.method == 'POST':
        form = FeedbackForm(request.POST)
        if form.is_valid():
            feedback = form.save(commit=False)
            feedback.user = request.user
            feedback.save()
            return redirect('dashboard')
    else:
        form = FeedbackForm()
    return render(request, 'customer/feedback.html', {'form': form})
```

URL Configuration (urls.py)

```
from django.urls import path
from . import views
urlpatterns = [
    path('', views.dashboard, name='dashboard'),
    path('add-courier/', views.add_courier, name='add_courier'),
    path('upload-image/', views.upload_image, name='upload_image'),
    path('feedback/', views.submit_feedback, name='submit_feedback'),
]
```

Templates

Admin Dashboard (admin/dashboard.html)

```
<h2>Admin Dashboard</h2>
<ul>
  <li><a href="/admin/courier/">Manage Couriers</a></li>
  <li><a href="/admin/feedback/">View Feedback</a></li>
  <li><a href="/admin/auth/user/">Manage Users</a></li>
</ul>
```

Employee Add Courier (employee/add_courier.html)

```
<h2>Add New Courier</h2>
<form method="post">
  {% csrf_token %}
  {{ form.as_p }}
  <button type="submit">Submit</button>
</form>
```

Customer Feedback (customer/feedback.html)

```
<h2>Submit Feedback</h2>
<form method="post">
  {% csrf_token %}
  {{ form.as_p }}
  <button type="submit">Send</button>
</form>
```

Media and Static Files Settings (settings.py)

```
MEDIA_URL = '/media/'
MEDIA_ROOT = BASE_DIR / 'media'
STATIC_URL = '/static/'
STATICFILES_DIRS = [BASE_DIR / "static"]
```

Admin Panel Configuration (admin.py)

```
from django.contrib import admin
from .models import Courier, Feedback, ImageUpload
admin.site.register(Courier)
admin.site.register(Feedback)
admin.site.register(ImageUpload)
```

Final Testing and Deployment

Test all modules (admin, employee, customer) for CRUD operations.

Use Django’s test client or Postman for API testing.

Deploy using services like PythonAnywhere or Render.

The implementation phase successfully transforms the system design into a working web application. By leveraging Django's ORM, user authentication, and MVC structure, combined with MySQL's reliability, this Courier Management System provides a secure and intuitive interface for all stakeholders—admin, employees, and customers.

4.2 SOFTWARE ENVIRONMENT

4.2.1 PYTHON

Python is a general-purpose interpreted, interactive, object-oriented, and high-level programming language. An interpreted language, Python has a design philosophy that emphasizes code readability (notably using whitespace indentation to delimit code blocks rather than curly brackets or keywords), and a syntax that allows programmers to express concepts in fewer lines of code than might be used in languages such as C++or Java. It provides constructs that enable clear programming on both small and large scales. Python interpreters are available for many operating systems. CPython, the reference implementation of Python, is open source software and has a community-based development model, as do nearly all of its variant implementations. CPython is managed by the non-profit Python Software Foundation. Python features a dynamic type system and automatic memory management. It supports multiple programming paradigms, including object-oriented, imperative, functional and procedural, and has a large and comprehensive standard library

What is Python

Python is a popular programming language. It was created by Guido van Rossum, and released in 1991.

It is used for:

- web development (server-side),
- software development,
- mathematics,
- system scripting.

What can Python do

- Python can be used on a server to create web applications.
- Python can be used alongside software to create workflows.
- Python can connect to database systems. It can also read and modify files.
- Python can be used to handle big data and perform complex mathematics.
- Python can be used for rapid prototyping, or for production-ready software development.

4.2.2 WHY CHOOSE PYTHON

Python has emerged as one of the most preferred programming languages for software development across a variety of domains, including machine learning, artificial intelligence, web development, and data science. The primary reason for choosing Python for the development of the Skin Care Product Recommendation System lies in its simplicity, readability, extensive library support, and vibrant community. Python offers a clean and straightforward syntax that closely resembles the English language, making it easy for developers to write and understand code efficiently. This simplicity accelerates the development process, reduces the chances of errors, and enhances maintainability, which is crucial for projects that involve multiple modules and complex workflows, such as machine learning pipelines and graphical user interface integration.

- Python works on different platforms (Windows, Mac, Linux, Raspberry Pi, etc).
- Python has a simple syntax similar to the English language.
- Python has syntax that allows developers to write programs with fewer lines than some other programming languages.
- Python runs on an interpreter system, meaning that code can be executed as soon as it is written. This means that prototyping can be very quick.
- Python can be treated in a procedural way, an object-orientated way or a functional way.

Good to know

- The most recent major version of Python is Python 3, which we shall be using in this tutorial. However, Python 2, although not being updated with anything other than security updates, is still quite popular.
- In this tutorial Python will be written in a text editor. It is possible to write Python in an Integrated Development Environment, such as Thonny, Pycharm, Netbeans or Eclipse which are particularly useful when managing larger collections of Python files.

Python Syntax compared to other programming languages

- Python was designed for readability, and has some similarities to the English language with influence from mathematics.
- Python uses new lines to complete a command, as opposed to other programming languages which often use semicolons or parentheses.
- Python relies on indentation, using whitespace, to define scope; such as the scope of loops, functions and classes. Other programming languages often use curly-brackets for this purpose.

History of Python:

Python is a fairly old language created by Guido Van Rossum. The design began in the late 1980s and was first released in February 1991.

Why Python was created?

In late 1980s, Guido Van Rossum was working on the Amoeba distributed operating system group. He wanted to use an interpreted language like ABC (ABC has simple easy-to-understand syntax) that could access the Amoeba system calls. So, he decided to create a language that was extensible. This led to design of a new language which was later named Python.

Why the name Python?

No. It wasn't named after a dangerous snake. Rossum was fan of a comedy series from late seventies. The name "Python" was adopted from the same series "Monty Python's Flying Circus".

Features of Python:

A simple language which is easier to learn

Python has a very simple and elegant syntax. It's much easier to read and write Python programs compared to other languages like: C++, Java, C#. Python makes programming fun and allows you to focus on the solution rather than syntax.

If you are a newbie, it's a great choice to start your journey with Python.

Free and open-source

You can freely use and distribute Python, even for commercial use. Not only can you use and distribute software’s written in it, you can even make changes to the Python's source code.

Python has a large community constantly improving it in each iteration.

Portability

You can move Python programs from one platform to another, and run it without any changes.

It runs seamlessly on almost all platforms including Windows, Mac OS X and Linux.

Extensible and Embeddable

Suppose an application requires high performance. You can easily combine pieces of C/C++ or other languages with Python code.

This will give your application high performance as well as scripting capabilities which other languages may not provide out of the box.

A high-level, interpreted language

Unlike C/C++, you don't have to worry about daunting tasks like memory management, garbage collection and so on.

Likewise, when you run Python code, it automatically converts your code to the language your computer understands. You don't need to worry about any lower-level operations.

Large standard libraries to solve common tasks

Python has a number of standard libraries which makes life of a programmer much easier since you don't have to write all the code yourself. For example: Need to connect MySQL database on a Web server? You can use MySQLdb library using import MySQLdb .

Standard libraries in Python are well tested and used by hundreds of people. So you can be sure that it won't break your application.

Object-oriented

Everything in Python is an object. Object oriented programming (OOP) helps you solve a complex problem intuitively.

With OOP, you are able to divide these complex problems into smaller sets by creating objects.

Applications of Python:

1. Simple Elegant Syntax

Programming in Python is fun. It's easier to understand and write Python code. Why? The syntax feels natural. Take this source code for an example:

```
a = 2
b = 3
sum = a + b
print(sum)
```

2. Not overly strict

You don't need to define the type of a variable in Python. Also, it's not necessary to add semicolon at the end of the statement.

Python enforces you to follow good practices (like proper indentation). These small things can make learning much easier for beginners.

3. Expressiveness of the language

Python allows you to write programs having greater functionality with fewer lines of code. Here's a link to the source code of Tic-tac-toe game with a graphical interface and a smart computer opponent in less than 500 lines of code. This is just an example. You will be amazed how much you can do with Python once you learn the basics.

4. Great Community and Support

Python has a large supporting community. There are numerous active forums online which can be handy if you are stuck.

Applications of AI

Gaming − AI plays important role for machine to think of large number of possible positions based on deep knowledge in strategic games. for example, chess,river crossing, N-queens problems and etc.

Natural Language Processing − Interact with the computer that understands natural language spoken by humans.

Expert Systems − Machine or software provide explanation and advice to the users.

Vision Systems − Systems understand, explain, and describe visual input on the computer.

Speech Recognition − There are some AI based speech recognition systems have ability to hear and express as sentences and understand their meanings while a person talks to it. For example Siri and Google assistant.

Handwriting Recognition − The handwriting recognition software reads the text written on paper and recognize the shapes of the letters and convert it into editable text.

Intelligent Robots − Robots are able to perform the instructions given by a human.

Major Goals

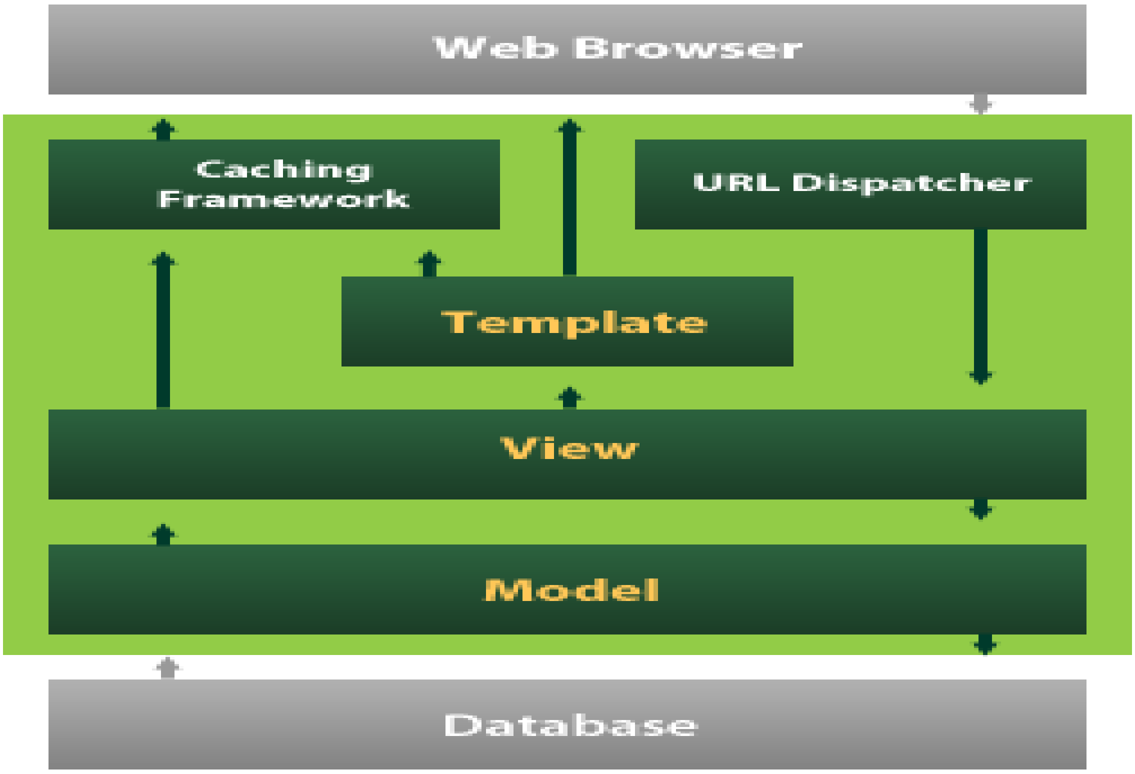
- Knowledge reasoning
- Planning

- Machine Learning
- Natural Language Processing
- Computer Vision
- Robotics

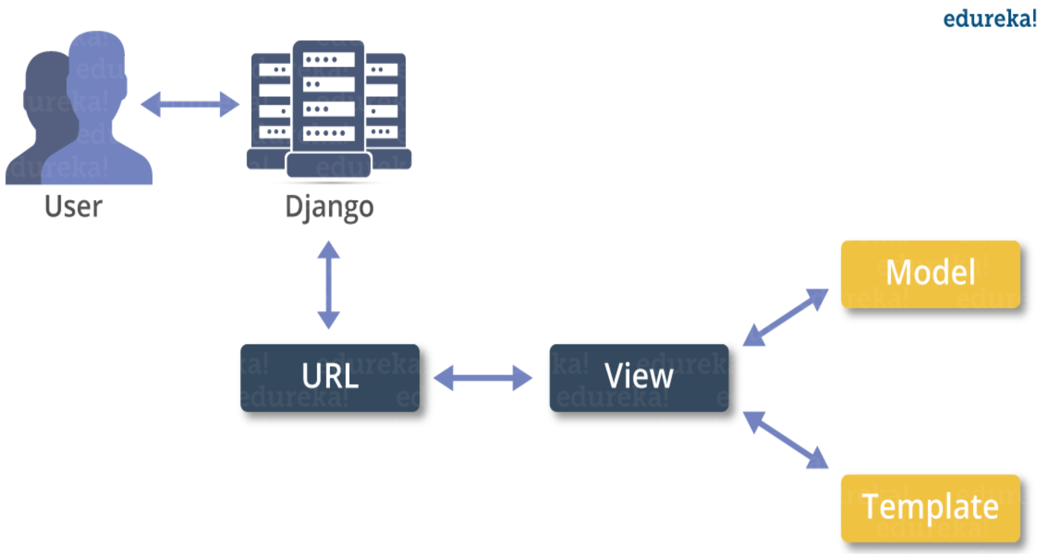
DJANGO

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's primary goal is to ease the creation of complex, database-driven websites. Django emphasizes reusability and "pluggability" of components, rapid development, and the principle of don't repeat yourself. Python is used throughout, even for settings files and data models.



Django also provides an optional administrative create, read, update and delete interface that is generated dynamically through introspection and configured via admin models



Purpose

The project involved analyzing the design of few applications so as to make the application more users friendly. To do so, it was really important to keep the navigations from one screen to the other well ordered and at the same time reducing the amount of typing the user needs to do. In order to make the application more accessible, the browser version had to be chosen so that it is compatible with most of the Browsers.

4.3 SAMPLE CODE

4.3.1 MAIN PROGRAM

```
from django.shortcuts import render
from django.template import RequestContext
from django.contrib import messages
from django.http import HttpResponse
import os
import pickle
import pymysql
import os
from django.core.files.storage import FileSystemStorage
from datetime import date
import matplotlib.pyplot as mplt
import io
import base64
import numpy as np
global uname
def ViewFeedback(request):
    if request.method == 'GET':
        output = '<table border=1><tr>'
        output+='<td><font size="" color="black">Courier&nbsp;ID</td>'
        output+='<td><font size="" color="black">Feedback</td>'
        output+='<td><font size="" color="black">Feedback Date</td>'
        output+='<td><font size="" color="black">Feedback Rank</td></tr>'
        rank = []
        con = pymysql.connect(host='127.0.0.1',port = 3306,user = 'root', password = 'root', database = 
'courier',charset='utf8')
        with con:
            cur = con.cursor()
            cur.execute("select * FROM feedback")
            rows = cur.fetchall()
            for row in rows:
                output+='<tr><td><font size="" color="black">'+str(row[0])+'</td>'
                output+='<td><font size="" color="black">'+str(row[1])+'</td>'
                output+='<td><font size="" color="black">'+str(row[2])+'</td>'
                output+='<td><font size="" color="black">'+str(row[3])+'</td></tr>'
                rank.append(row[3])
        output += "</table><br/>"
        unique, count = np.unique(np.asarray(rank), return_counts=True)
        mplt.pie(count,labels=unique,autopct='%1.1f%%')
        mplt.title('Feedback Ranking Graph')
        mplt.axis('equal')
        buf = io.BytesIO()
        mplt.savefig(buf, format='png', bbox_inches='tight')
        mplt.close()
        img_b64 = base64.b64encode(buf.getvalue()).decode()
        context= {'data':output, 'img': img_b64}
        return render(request, 'ViewFeedback.html', context)
def getDetails(pid):
    status = False
    emp = ""
    sender = ""
    sender_phone = ""
    receiver = ""
    amount= ""
    delivery = ""
    img = ""
    con = pymysql.connect(host='127.0.0.1',port = 3306,user = 'root', password = 'root', database =
'courier',charset='utf8')
    with con:
        cur = con.cursor()
        cur.execute("select
collected_employee,sender_name,sender_phone,receiver_name,amount,expected_delivery,parcel_im
age FROM parcel where parcel_id = '"+pid+"'")
        rows = cur.fetchall()
        for row in rows:
            emp = row[0]
            sender = row[1]
            sender_phone = row[2]
            receiver = row[3]
            amount = row[4]
            delivery = row[5]
            img = row[6]
            break
    return emp,sender,sender_phone,receiver,amount,delivery,img
def UserMap(request):
    if request.method == 'GET':
        name = request.GET.get('t1', False)
        output = '<iframe width="625" height="650" frameborder="0" scrolling="no" marginheight="0"
marginwidth="0"
src="https://maps.google.com/maps?q='+name+'&amp;ie=UTF8&amp;&amp;output=embed"></ifra
me><br/>'
        context= {'data':output}
        return render(request, 'UserMap.html', context)

def CourierTrack(request):
    if request.method == 'GET':
       return render(request, 'CourierTrack.html', {})
```

```
def CourierTrackAction(request):
    if request.method == 'POST':
        global uname
        pid = request.POST.get('t1', False)
        emp,sender,sender_phone,receiver,amount,delivery,img = getDetails(pid)
        output = ''
        output+='<table border=1 align=center width=100%><tr><th><font size="" 
color="black">Collected Employee</th><th><font size="" color="black">Sender Name</th>'
        output+='<th><font size="" color="black">Sender Phone</th><th><font size="" 
color="black">Receiver Name</th>'
        output+='<th><font size="" color="black">Amount</th><th><font size="" 
color="black">Expected Delivery</th>'
        output+='<th><font size="" color="black">Current Location</th>'
        output+='<th><font size="" color="black">Updated Date</th><th><font size="" 
color="black">Status</th>'
        output += '<th><font size="" color="black">Parcel Image</th><th><font size="" 
color="black">View on Map</th></tr>'
        con = pymysql.connect(host='127.0.0.1',port = 3306,user = 'root', password = 'root', database = 
'courier',charset='utf8')
        with con:
            cur = con.cursor()
            cur.execute("select * FROM update_status where parcel_id = '"+pid+"'")
            rows = cur.fetchall()
            for row in rows:
                output+='<td><font size="" color="black">'+emp+'</td><td><font size="" 
color="black">'+sender+'</td>'
                output+='<td><font size="" color="black">'+sender_phone+'</td><td><font size="" 
color="black">'+receiver+'</td>'
                output+='<td><font size="" color="black">'+amount+'</td><td><font size="" 
color="black">'+delivery+'</td>'
                output+='<td><font size="" color="black">'+row[2]+'</td>'
                output+='<td><font size="" color="black">'+row[3]+'</td>'
                output+='<td><font size="" color="black">'+row[4]+'</td>'
                output+='<td><img src="/static/files/'+img+'" width="200" height="200"></img></td>'
            output+='<td><a href=\'UserMap?t1='+str(row[2])+'\'><font size=3 color=black>View on Map</font></a></td></tr>'
        output+= "</table></br></br></br></br>"
        context= {'data':output}
        return render(request, 'UserMap.html', context)

def EmployeeMap(request):
    if request.method == 'GET':
        name = request.GET.get('t1', False)
        output = '<iframe width="625" height="650" frameborder="0" scrolling="no" marginheight="0" marginwidth="0" src="https://maps.google.com/maps?q='+name+'&amp;ie=UTF8&amp;&amp;output=embed"></iframe><br/>'
        context= {'data':output}
        return render(request, 'EmployeeMap.html', context)

def ViewCurrentStatusAction(request):
    if request.method == 'POST':
        global uname
        pid = request.POST.get('t1', False)
        emp,sender,sender_phone,receiver,amount,delivery,img = getDetails(pid)
        output = ''
        output+='<table border=1 align=center width=100%><tr><th><font size="" color="black">Collected Employee</th><th><font size="" color="black">Sender Name</th>'
        output+='<th><font size="" color="black">Sender Phone</th><th><font size="" color="black">Receiver Name</th>'
        output+='<th><font size="" color="black">Amount</th><th><font size="" color="black">Expected Delivery</th>'
        output+='<th><font size="" color="black">Current Location</th>'
        output+='<th><font size="" color="black">Updated Date</th><th><font size="" color="black">Status</th>'
        output += '<th><font size="" color="black">Parcel Image</th><th><font size="" color="black">View on Map</th></tr>'
        con = pymysql.connect(host='127.0.0.1',port = 3306,user = 'root', password = 'root', database = 'courier',charset='utf8')
        with con:
            cur = con.cursor()
            cur.execute("select * FROM update_status where parcel_id = '"+pid+"'")
            rows = cur.fetchall()
            for row in rows:
            output+='<td><font size="" color="black">'+emp+'</td><td><font size="" color="black">'+sender+'</td>'
            output+='<td><font size="" color="black">'+sender_phone+'</td><td><font size="" color="black">'+receiver+'</td>'
            output+='<td><font size="" color="black">'+amount+'</td><td><font size="" color="black">'+delivery+'</td>'
            output+='<td><font size="" color="black">'+row[2]+'</td>'
            output+='<td><font size="" color="black">'+row[3]+'</td>'
            output+='<td><font size="" color="black">'+row[4]+'</td>'
            output+='<td><img src="/static/files/'+img+'" width="200" height="200"></img></td>'
            output+='<td><a href=\'EmployeeMap?t1='+str(row[2])+'\'><font size=3 color=black>View on Map</font></a></td></tr>'
        output+= "</table></br></br></br></br>"
        context= {'data':output}
        return render(request, 'EmployeeScreen.html', context)


def ViewCurrentStatus(request):
    if request.method == 'GET':
        output = '<tr><td><font size=" color="black"><b>Courier&nbsp;ID</b></td><td><select name="t1">'
        con = pymysql.connect(host='127.0.0.1',port = 3306,user = 'root', password = 'root', database = 'courier',charset='utf8')
        with con:
            cur = con.cursor()
            cur.execute("select parcel_id FROM parcel")
            rows = cur.fetchall()
            for row in rows:
            pid = str(row[0])
            output += '<option value="'+str(pid)+'">'+str(pid)+'</option>'
        context= {'data1': output}
        return render(request, 'ViewCurrentStatus.html', context)

def checkStatus(pid):
    status = False
    con = pymysql.connect(host='127.0.0.1',port = 3306,user = 'root', password = 'root', database =
'courier',charset='utf8')
    with con:
        cur = con.cursor()
        cur.execute("select * FROM update_status where status = 'Delivered' and parcel_id='"+pid+"'")
        rows = cur.fetchall()
        for row in rows:
            status = True
            break
    return status

def UpdateCourier(request):
    if request.method == 'GET':
        output = '<tr><td><font size="' color="black"><b>Courier&nbsp;ID</b></td><td><select
name="t1">'
        con = pymysql.connect(host='127.0.0.1',port = 3306,user = 'root', password = 'root', database =
'courier',charset='utf8')
        with con:
            cur = con.cursor()
            cur.execute("select parcel_id FROM parcel")
            rows = cur.fetchall()
            for row in rows:
                pid = str(row[0])
                status = checkStatus(pid)
                if status == False:
                    output += '<option value="'+str(pid)+'">'+str(pid)+'</option>'
        context= {'data1': output}
        return render(request, 'UpdateCourier.html', context)

def UpdateCourierAction(request):
    if request.method == 'POST':
        global uname
        pid = request.POST.get('t1', False)
        location = request.POST.get('t2', False)
        status = request.POST.get('t3', False)
        today = str(date.today())
        db_connection = pymysql.connect(host='127.0.0.1',port = 3306,user = 'root', password = 'root',
database = 'courier',charset='utf8')
        db_cursor = db_connection.cursor()
        student_sql_query = "INSERT INTO
update_status(parcel_id,employee_reporting,current_location,update_date,status)
VALUES('"+str(pid)+"','"+uname+"','"+location+"','"+str(today)+"','"+status+"')"
        db_cursor.execute(student_sql_query)
        db_connection.commit()
        print(db_cursor.rowcount, "Record Inserted")
        if db_cursor.rowcount == 1:
            status = "Courier status updated successfully"
        context= {'data': status}
        return render(request, 'EmployeeScreen.html', context)

def Feedback(request):
    if request.method == 'GET':
       return render(request, 'Feedback.html', {})

def FeedbackAction(request):
    if request.method == 'POST':
        global uname
        pid = request.POST.get('t1', False)
        feedback = request.POST.get('t2', False)
        rating = request.POST.get('t3', False)
        today = str(date.today())
        db_connection = pymysql.connect(host='127.0.0.1',port = 3306,user = 'root', password = 'root',
database = 'courier',charset='utf8')
        db_cursor = db_connection.cursor()
        student_sql_query = "INSERT INTO
feedback(username,feedback,feedback_date,feedback_rank)
VALUES('"+str(pid)+"','"+feedback+"','"+str(today)+"','"+rating+"')"
        db_cursor.execute(student_sql_query)
        db_connection.commit()
        print(db_cursor.rowcount, "Record Inserted")
        if db_cursor.rowcount == 1:
            status = "Your Feedback Accepted. Our Admin will review"
        context= {'data': status}
        return render(request, 'Feedback.html', context)

def CollectCourierAction(request):
    if request.method == 'POST':
        global uname
        sender = request.POST.get('t1', False)
        sender_phone = request.POST.get('t2', False)
        sender_address = request.POST.get('t3', False)
        receiver = request.POST.get('t4', False)
        receiver_phone = request.POST.get('t5', False)
        receiver_address = request.POST.get('t6', False)
        desc = request.POST.get('t7', False)
        weight = request.POST.get('t8', False)
        amount = request.POST.get('t9', False)
        delivery = request.POST.get('t10', False)
        image = request.FILES['t11']
        imagename = request.FILES['t11'].name
        status = "none"
        pid = 1
        con = pymysql.connect(host='127.0.0.1',port = 3306,user = 'root', password = 'root', database =
'courier',charset='utf8')
        with con:
            cur = con.cursor()
            cur.execute("select max(parcel_id) FROM parcel")
            rows = cur.fetchall()
            for row in rows:
                pid = row[0]
        if pid is not None:
            pid += 1
        else:
            pid = 1
        today = str(date.today())
        db_connection = pymysql.connect(host='127.0.0.1',port = 3306,user = 'root', password = 'root', database = 'courier',charset='utf8')
        db_cursor = db_connection.cursor()
        student_sql_query = "INSERT INTO parcel(parcel_id,collected_employee,sender_name,sender_phone,sender_address,receiver_name,receiver_phone,receiver_address,description,parcel_weight,amount,collected_date,expected_delivery,parcel_image) VALUES('"+str(pid)+"','"+uname+"','"+sender+"','"+sender_phone+"','"+sender_address+"','"+receiver+"','"+receiver_phone+"','"+receiver_address+"','"+desc+"','"+weight+"','"+amount+"','"+str(today)+"','"+delivery+"','"+imagename+"')"
        db_cursor.execute(student_sql_query)
        db_connection.commit()
        print(db_cursor.rowcount, "Record Inserted")
        if db_cursor.rowcount == 1:
            fs = FileSystemStorage()
            if os.path.exists('CourierApp/static/files/'+imagename):
                os.remove('CourierApp/static/files/'+imagename)
            filename = fs.save('CourierApp/static/files/'+imagename, image)
            status = "Courier details added<br/>Courier Tracking ID : "+str(pid)
        context= {'data': status}
        return render(request, 'CollectCourier.html', context)
def CollectCourier(request):
    if request.method == 'GET':
       return render(request, 'CollectCourier.html', {})
def AdminLogin(request):
    if request.method == 'GET':
       return render(request, 'AdminLogin.html', {})

def EmployeeLogin(request):
    if request.method == 'GET':
       return render(request, 'EmployeeLogin.html', {})
def CourierTrack(request):
    if request.method == 'GET':
       return render(request, 'CourierTrack.html', {})
def AddEmployee(request):
    if request.method == 'GET':
       return render(request, 'AddEmployee.html', {})
def index(request):
    if request.method == 'GET':
       return render(request, 'index.html', {})
def AdminLoginAction(request):
    if request.method == 'POST':
        global uname
        username = request.POST.get('t1', False)
        password = request.POST.get('t2', False)
        if username == 'admin' and password == 'admin':
            context= {'data':'welcome '+username}
            return render(request, 'AdminScreen.html', context)
        else:
            context= {'data':'login failed'}
            return render(request, 'AdminLogin.html', context)
def EmployeeLoginAction(request):
    if request.method == 'POST':
        global uname
        username = request.POST.get('t1', False)
        password = request.POST.get('t2', False)
        index = 0
        con = pymysql.connect(host='127.0.0.1',port = 3306,user = 'root', password = 'root', database =
'courier',charset='utf8')
        with con:
            cur = con.cursor()
            cur.execute("select username, password FROM employee")
            rows = cur.fetchall()
            for row in rows:
                if row[0] == username and password == row[1]:
                    uname = username
                    index = 1
                    break
        if index == 1:
            context= {'data':'welcome '+username}
            return render(request, 'EmployeeScreen.html', context)
        else:
            context= {'data':'login failed'}
            return render(request, 'EmployeeLogin.html', context)
    def AddEmployeeAction(request):
    if request.method == 'POST':
        name = request.POST.get('t1', False)
        gender = request.POST.get('t2', False)
        contact = request.POST.get('t3', False)
        email = request.POST.get('t4', False)
        qualification = request.POST.get('t5', False)
        experience = request.POST.get('t6', False)
        address = request.POST.get('t7', False)
        username = request.POST.get('t8', False)
        password = request.POST.get('t9', False)
        status = "none"
        con = pymysql.connect(host='127.0.0.1',port = 3306,user = 'root', password = 'root', database =
'courier',charset='utf8')
        with con:
            cur = con.cursor()
            cur.execute("select username FROM employee")
            rows = cur.fetchall()
            for row in rows:
                if row[0] == username:
                    status = "Username already exists"
                    break
        if status == "none":
            db_connection = pymysql.connect(host='127.0.0.1',port = 3306,user = 'root', password = 'root',
database = 'courier',charset='utf8')
```

```
            db_cursor = db_connection.cursor()
            student_sql_query = "INSERT INTO
employee(employee_name,gender,contact_no,email,qualification,experience,address,username,pass
word)
VALUES('"+name+"','"+gender+"','"+contact+"','"+email+"','"+qualification+"','"+experience+"','"+
address+"','"+username+"','"+password+"')"
            db_cursor.execute(student_sql_query)
            db_connection.commit()
            print(db_cursor.rowcount, "Record Inserted")
            if db_cursor.rowcount == 1:
                status = "Employee details added<br/>Employee can login with "+username
        context= {'data': status}
        return render(request, 'AddEmployee.html', context)

def ViewEmployees(request):
    if request.method == 'GET':
        output = ''
        output+='<table border=1 align=center width=100%><tr><th><font size=""
color="black">Employee Name</th><th><font size="" color="black">Gender</th>'
        output+='<th><font size="" color="black">Contact No</th><th><font size=""
color="black">Email ID</th>'
        output+='<th><font size="" color="black">Qualification</th><th><font size=""
color="black">Experience</th>'
        output+='<th><font size="" color="black">Address</th>'
        output+='<th><font size="" color="black">Username</th><th><font size=""
color="black">Password</th></tr>'
        con = pymysql.connect(host='127.0.0.1',port = 3306,user = 'root', password = 'root', database =
'courier',charset='utf8')
        with con:
            cur = con.cursor()
            cur.execute("select * from employee")
            rows = cur.fetchall()
            output+='<tr>'
            for row in rows:
                output+='<td><font size="" color="black">'+row[0]+'</td><td><font size=""
color="black">'+str(row[1])+'</td>'
                output+='<td><font size="" color="black">'+row[2]+'</td><td><font size=""
color="black">'+row[3]+'</td>'
                output+='<td><font size="" color="black">'+row[4]+'</td><td><font size=""
color="black">'+row[5]+'</td>'
                output+='<td><font size="" color="black">'+row[6]+'</td>'
                output+='<td><font size="" color="black">'+row[7]+'</td>'
                output+='<td><font size="" color="black">'+row[8]+'</td></tr>'
        output+= "</table></br></br></br></br>"
        context= {'data':output}
        return render(request, 'AdminScreen.html', context)

def ViewCouriers(request):
    if request.method == 'GET':
        output = ''
        output+='<table border=1 align=center width=100%><tr><th><font size=""
color="black">Parcel ID</th><th><font size="" color="black">Collected Employee</th>'
        output+='<th><font size="" color="black">Sender Name</th><th><font size=""
color="black">Sender Phone</th>'
        output+='<th><font size="" color="black">Receiver Name</th><th><font size=""
color="black">Receiver Phone</th>'
        output+='<th><font size="" color="black">Description</th>'
        output+='<th><font size="" color="black">Amount</th><th><font size=""
color="black">Collected Date</th>'
        output += '<th><font size="" color="black">Expected Delivery</th><th><font size=""
color="black">Parcel Image</th></tr>'
        con = pymysql.connect(host='127.0.0.1',port = 3306,user = 'root', password = 'root', database =
'courier',charset='utf8')
        with con:
            cur = con.cursor()
            cur.execute("select * from parcel")
            rows = cur.fetchall()
            output+='<tr>'
            for row in rows:
                output+='<td><font size="" color="black">'+str(row[0])+'</td><td><font size=""
color="black">'+str(row[1])+'</td>'
                output+='<td><font size="" color="black">'+row[2]+'</td><td><font size=""
color="black">'+row[3]+'</td>'
                output+='<td><font size="" color="black">'+row[5]+'</td><td><font size=""
color="black">'+row[6]+'</td>'
                output+='<td><font size="" color="black">'+row[8]+'</td>'
                output+='<td><font size="" color="black">'+row[10]+'</td><td><font size=""
color="black">'+row[11]+'</td>'
                output+='<td><font size="" color="black">'+row[12]+'</td>'
                output+='<td><img src="/static/files/'+row[13]+'" width="200"
height="200"></img></td></tr>'
        output+= "</table></br></br></br></br>"
        context= {'data':output}
        return render(request, 'AdminScreen.html', context)
```

SYSTEM TESTING AND SCREENSHOTS

5.SYSTEM TESTING AND SCREENSHORTS

5.1 SYSTEM TESTING

TESTING METHODOLOGIES

The purpose of testing is to discover errors. Testing is the process of trying to discover every conceivable fault or weakness in a work product. It provides a way to check the functionality of components, sub assemblies, assemblies and/or a finished product It is the process of exercising software with the intent of ensuring that the Software system meets its requirements and user expectations and does not fail in an unacceptable manner. There are various types of test. Each test type addresses a specific testing requirement.

TYPES OF TESTS

5.1.1 UNIT TSTING

Unit testing involves the design of test cases that validate that the internal program logic is functioning properly, and that program inputs produce valid outputs. All decision branches and internal code flow should be validated. It is the testing of individual software units of the application .it is done after the completion of an individual unit before integration. This is a structural testing, that relies on knowledge of its construction and is invasive. Unit tests perform basic tests at component level and test a specific business process, application, and/or system configuration. Unit tests ensure that each unique path of a business process performs accurately to the documented specifications and contains clearly defined inputs and expected results.

5.1.2 INTEGRATION TESTING

Integration tests are designed to test integrated software components to determine if they actually run as one program. Testing is event driven and is more concerned with the basic outcome of screens or fields. Integration tests demonstrate that although the components were individually satisfaction, as shown by successfully unit testing, the combination of components is correct and consistent. Integration testing is specifically aimed at exposing the problems that arise from the combination of components.

5.1.3 FUNCTIONAL TESTING

Functional tests provide systematic demonstrations that functions tested are available as specified by the business and technical requirements, system documentation, and user manuals.

Functional testing is centered on the following items:

Valid Input : identified classes of valid input must be accepted.

Invalid Input : identified classes of invalid input must be rejected.

Functions : identified functions must be exercised.

Output : identified classes of application outputs must be exercised.

Systems/Procedures : interfacing systems or procedures must be invoked.

Organization and preparation of functional tests is focused on requirements, key functions, or special test cases. In addition, systematic coverage pertaining to identify Business process flows; data fields, predefined processes, and successive processes must be considered for testing. Before functional testing is complete, additional tests are identified and the effective value of current tests is determined.

5.1.4 SYSTEM TESTING

System testing ensures that the entire integrated software system meets requirements. It tests a configuration to ensure known and predictable results. An example of system testing is the configuration oriented system integration test. System testing is based on process descriptions and flows, emphasizing pre-driven process links and integration points.

5.1.5 WHITE BOX TESTING

White Box Testing is a testing in which in which the software tester has knowledge of the inner workings, structure and language of the software, or at least its purpose. It is purpose. It is used to test areas that cannot be reached from a black box level.

5.1.6 BLACKBOX TESTING

Black Box Testing is testing the software without any knowledge of the inner workings, structure or language of the module being tested. Black box tests, as most other kinds of tests, must be written from a definitive source document, such as specification or requirements document, such as specification or requirements document. It is a testing in which the software under test is treated, as a black box .you cannot “see” into it. The test provides inputs and responds to outputs without considering how the software works.

UNIT TESTING

Unit testing is usually conducted as part of a combined code and unit test phase of the software lifecycle, although it is not uncommon for coding and unit testing to be conducted as two distinct phases.

Test strategy and approach

Field testing will be performed manually and functional tests will be written in detail.

Test objectives

All field entries must work properly.

Pages must be activated from the identified link.

The entry screen, messages and responses must not be delayed.

Features to be tested

Verify that the entries are of the correct format

No duplicate entries should be allowed

All links should take the user to the correct page.

INTEGRATION TESTING

Software integration testing is the incremental integration testing of two or more integrated software components on a single platform to produce failures caused by interface defects.

The task of the integration test is to check that components or software applications, e.g. components in a software system or – one step up – software applications at the company level – interact without error.

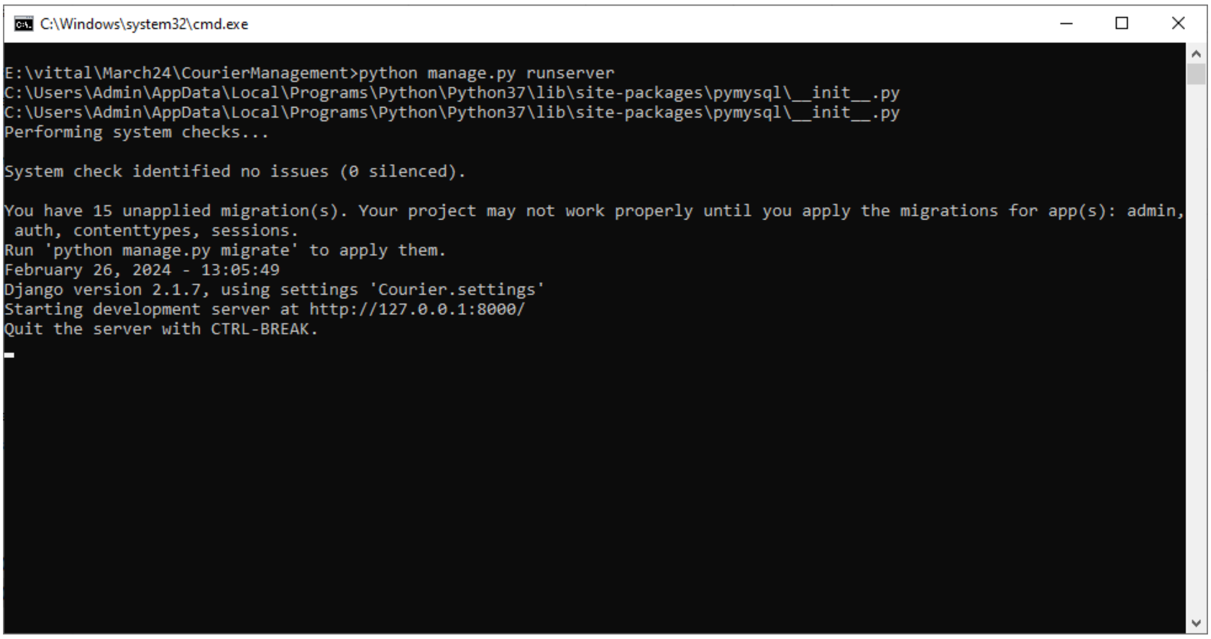
Test Results: All the test cases mentioned above passed successfully. No defects encountered.

5.1.7 ACCEPTANCE TESTING

User Acceptance Testing is a critical phase of any project and requires significant participation by the end user. It also ensures that the system meets the functional requirements.

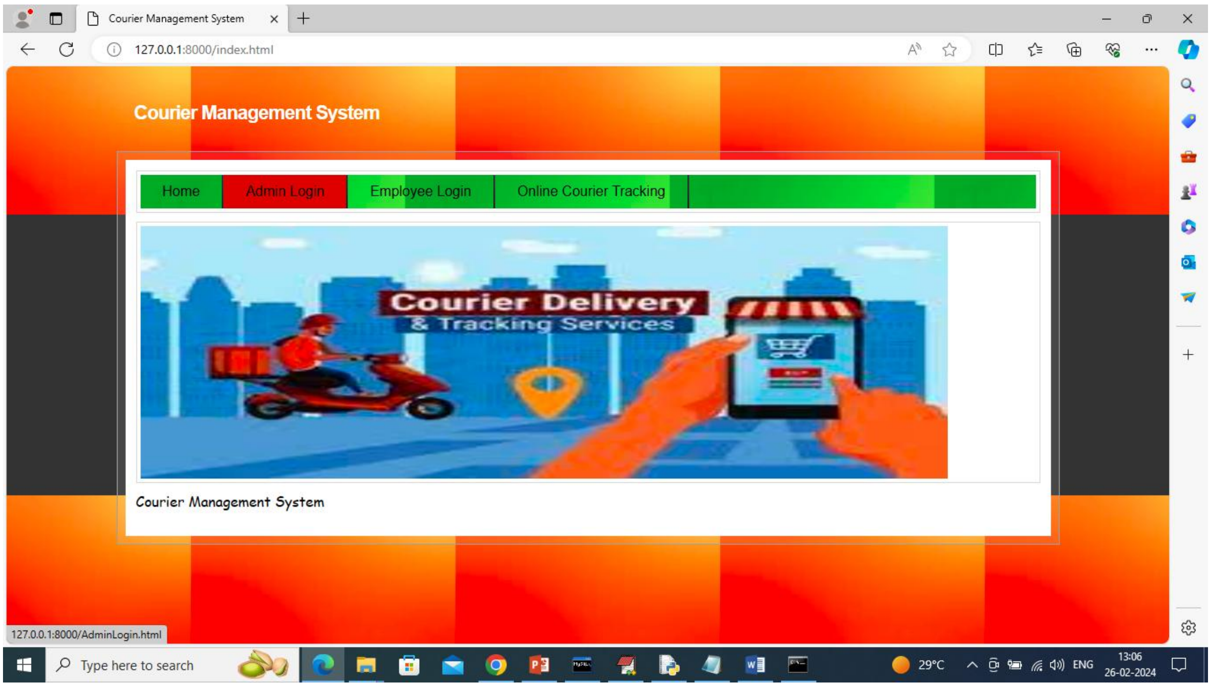
Test Results: All the test cases mentioned above passed successfully. No defects encountered.

5.2 SCREENSHOTS

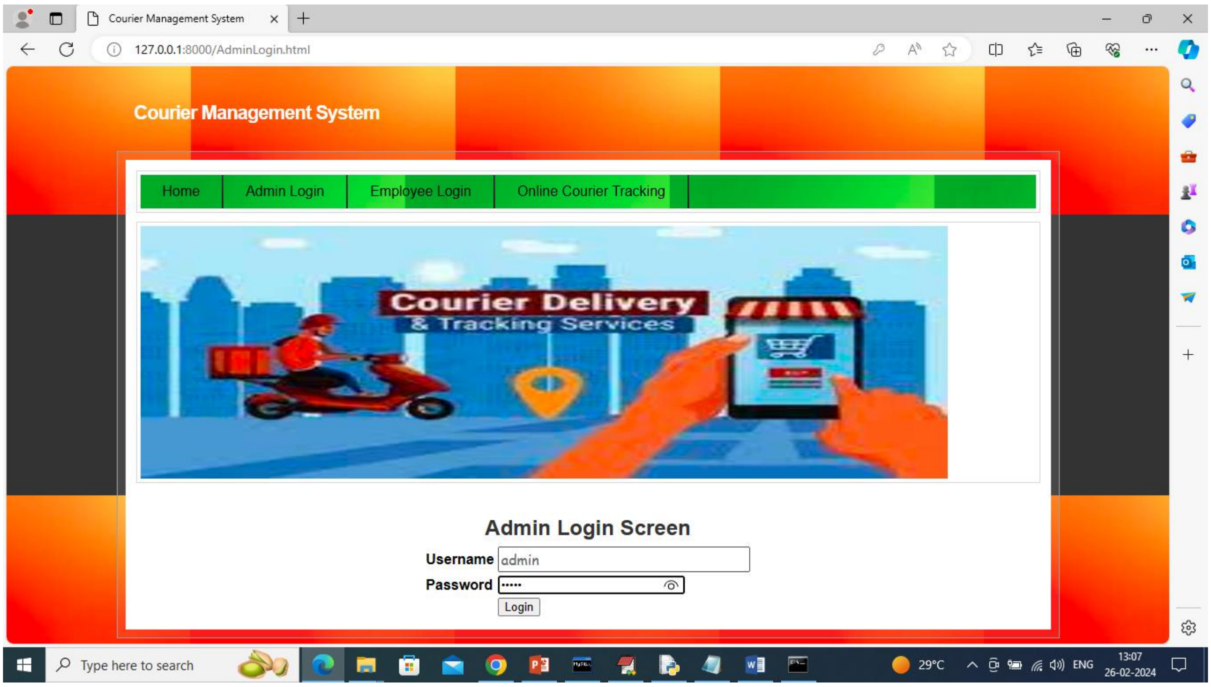


```
E:\vittal\March24\CourierManagement>python manage.py runserver
C:\Users\Admin\AppData\Local\Programs\Python\Python37\lib\site-packages\pymysql\__init__.py
C:\Users\Admin\AppData\Local\Programs\Python\Python37\lib\site-packages\pymysql\__init__.py
Performing system checks...

System check identified no issues (0 silenced).

You have 15 unapplied migration(s). Your project may not work properly until you apply the migrations for app(s): admin,
 auth, contenttypes, sessions.
Run 'python manage.py migrate' to apply them.
February 26, 2024 - 13:05:49
Django version 2.1.7, using settings 'Courier.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.
```

In above screen python server started and now open browser and enter URL as http://127.0.0.1:8000/index.html and press enter key to get below page.


Courier Management System
127.0.0.1:8000/index.html
Courier Management System
Home
Admin Login
Employee Login
Online Courier Tracking
Courier Delivery
& Tracking Services
Courier Management System
127.0.0.1:8000/AdminLogin.html
Type here to search
29°C
ENG
13:06
26-02-2024

In above screen click on ‘Admin Login’ link to get below login page



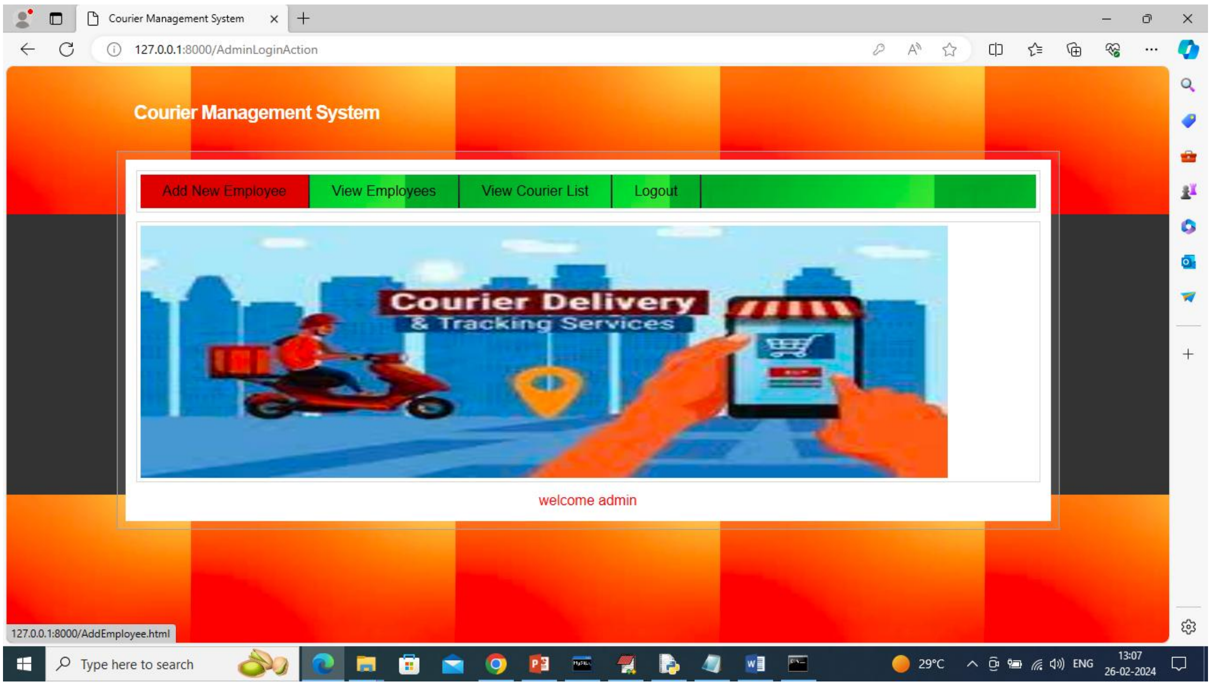
Admin Login Screen

Username admin

Password •••••

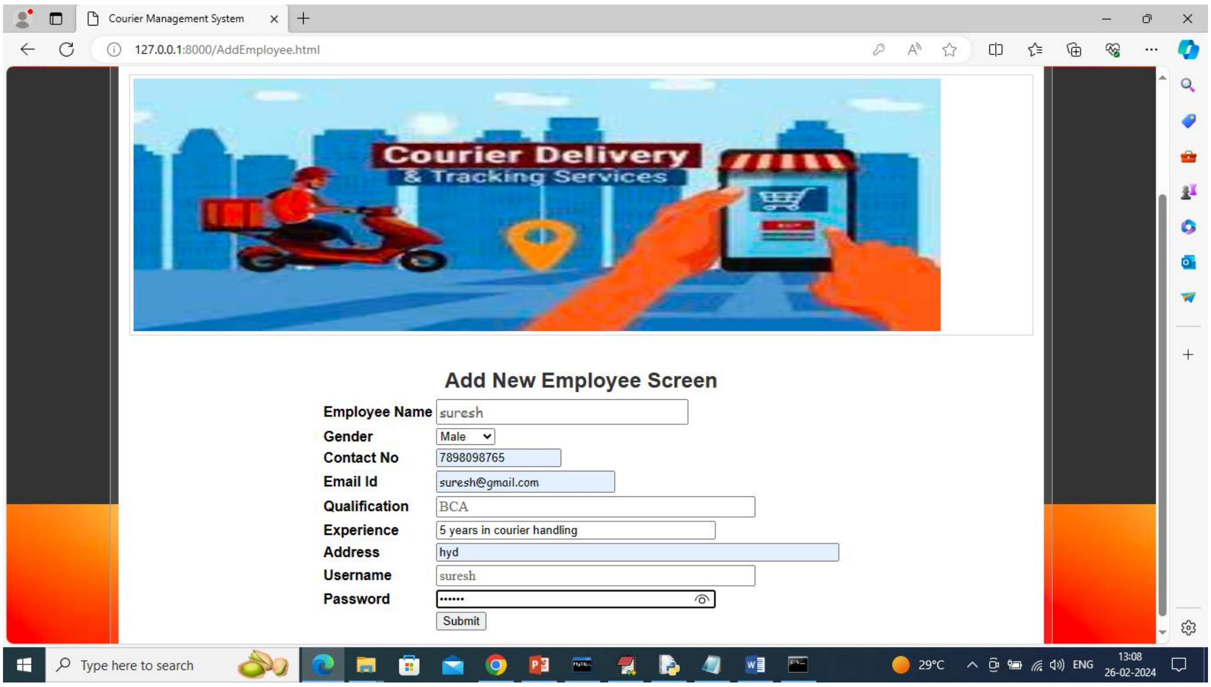
Login

In above screen click on ‘Admin Login’ link to get below login page



welcome admin

In above screen admin can click on ‘Add New Employee’ link to add employe details and get below page



Add New Employee Screen

Employee Name suresh

Gender Male

Contact No 7898098765

Email Id suresh@gmail.com

Qualification BCA

Experience 5 years in courier handling

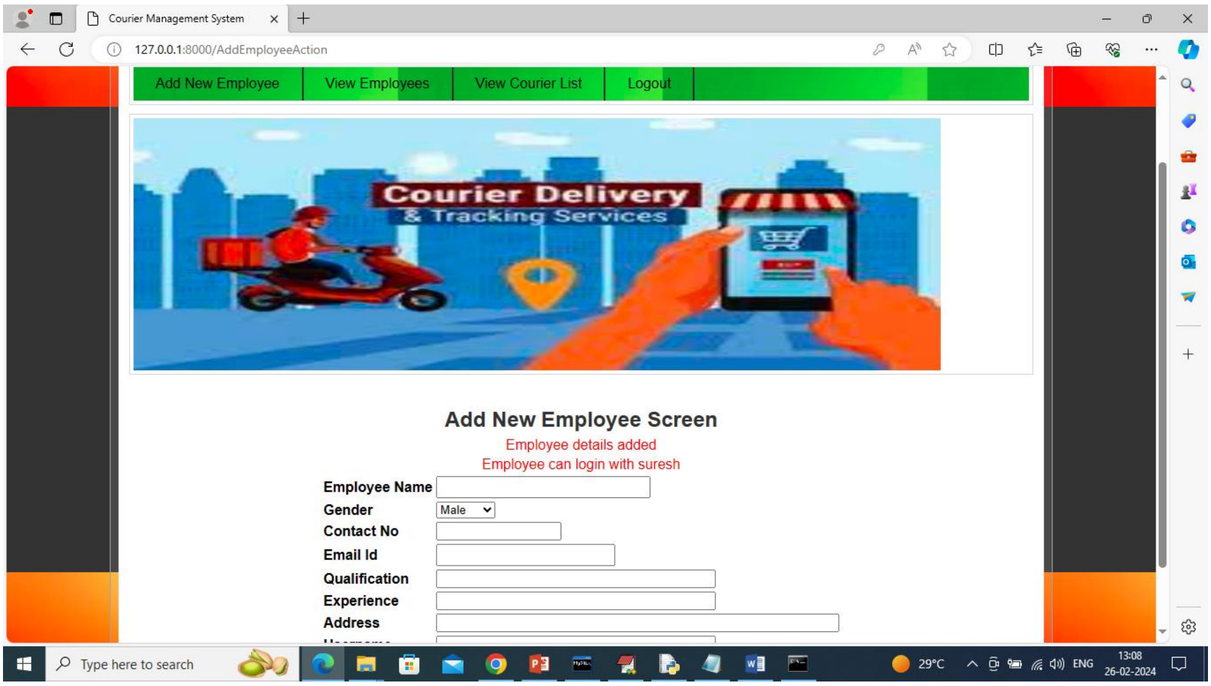
Address hyd

Username suresh

Password ••••••

Submit

In above screen admin will add employee details and then press button to save details



Add New Employee Screen

Employee details added

Employee can login with suresh

Employee Name

Gender Male

Contact No

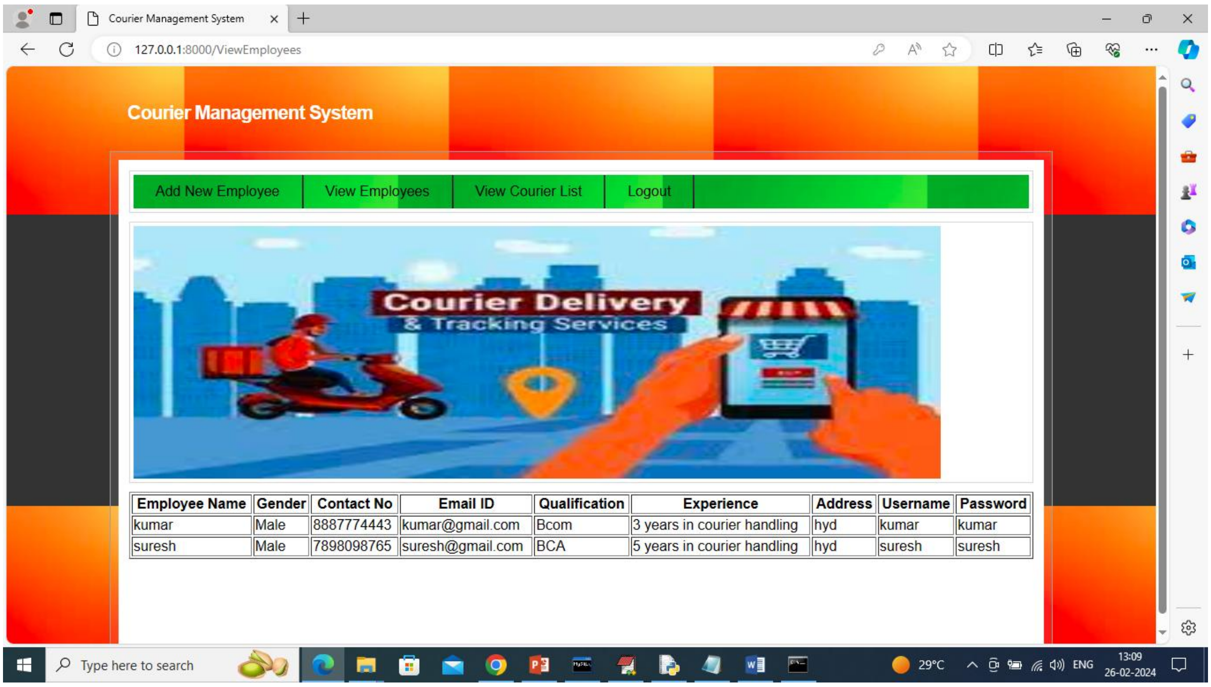
Email Id

Qualification

Experience

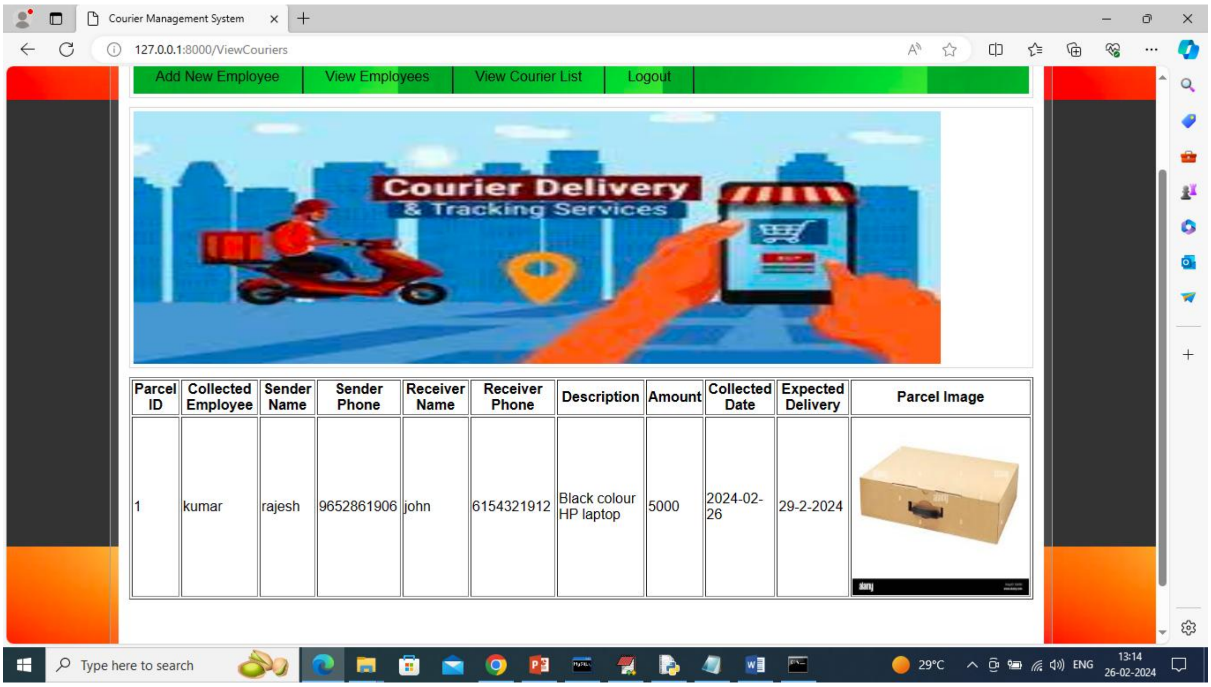
Address

In above screen employee details added to database and by using this login details employee can login to application and now click on ‘View Employee’ link to get list of all available employees



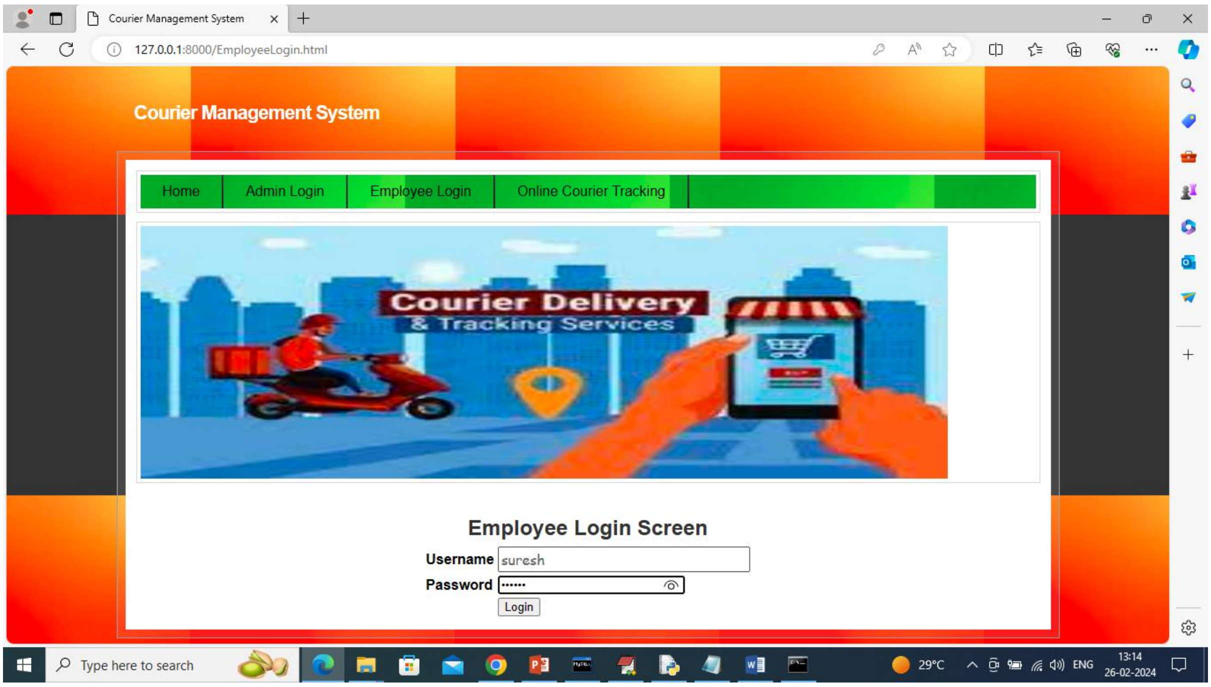
Employee Name	Gender	Contact No	Email ID	Qualification	Experience	Address	Username	Password
kumar	Male	8887774443	kumar@gmail.com	Bcom	3 years in courier handling	hyd	kumar	kumar
suresh	Male	7898098765	suresh@gmail.com	BCA	5 years in courier handling	hyd	suresh	suresh

In above screen admin can details of all employees and now click on ‘View Courier List’ to view details of all booked couriers



Parcel ID	Collected Employee	Sender Name	Sender Phone	Receiver Name	Receiver Phone	Description	Amount	Collected Date	Expected Delivery	Parcel Image
1	kumar	rajesh	9652861906	john	6154321912	Black colour HP laptop	5000	2024-02-26	29-2-2024	

In above screen admin can view list of all booked couriers list and now logout and login as employee

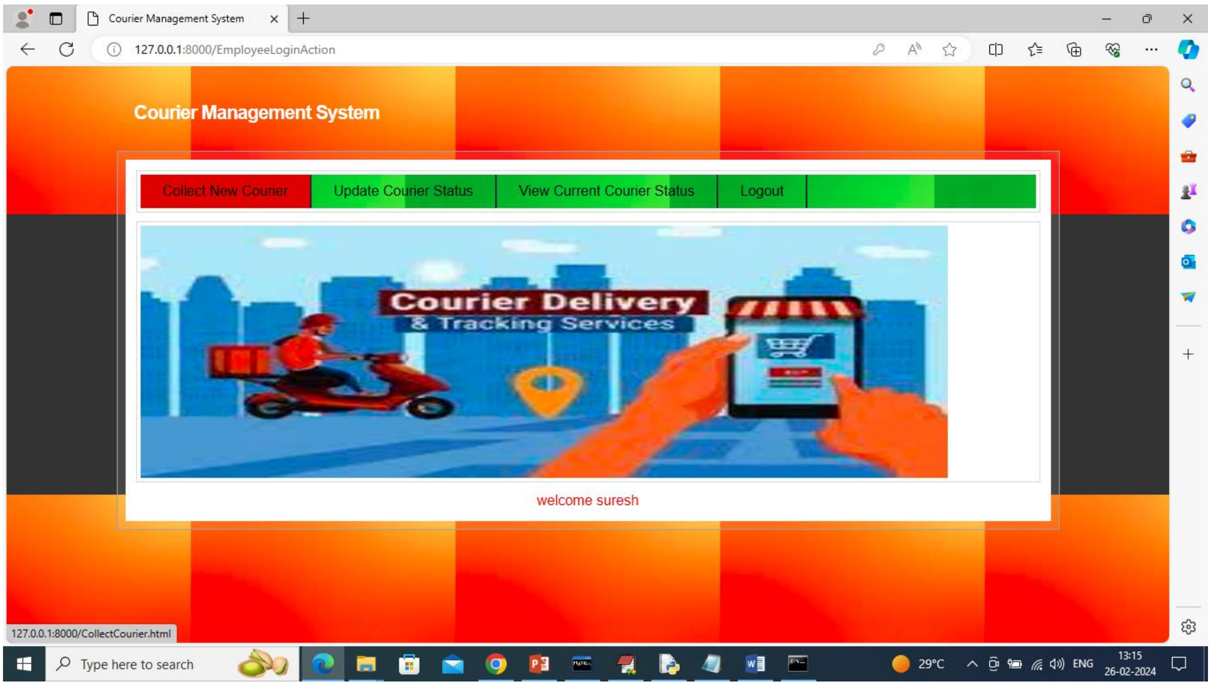


Employee Login Screen

Username suresh

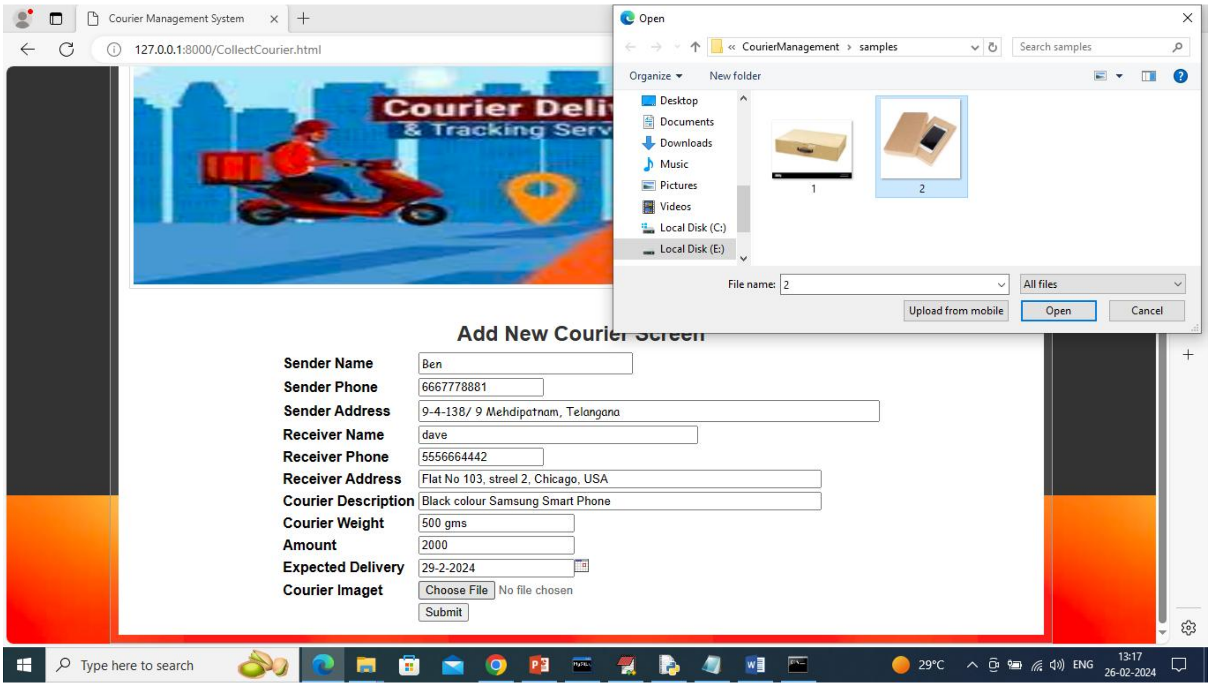
Password ••••••

In above screen employee is login and after login will get below page


Courier Management System
127.0.0.1:8000/EmployeeLoginAction
Courier Management System
Collect New Courier
Update Courier Status
View Current Courier Status
Logout
Courier Delivery
& Tracking Services
127.0.0.1:8000/CollectCourier.html
Type here to search
29°C
ENG
13:15
26-02-2024

welcome suresh

In above screen employee will click on ‘Collect New Courier’ link to add new courier details

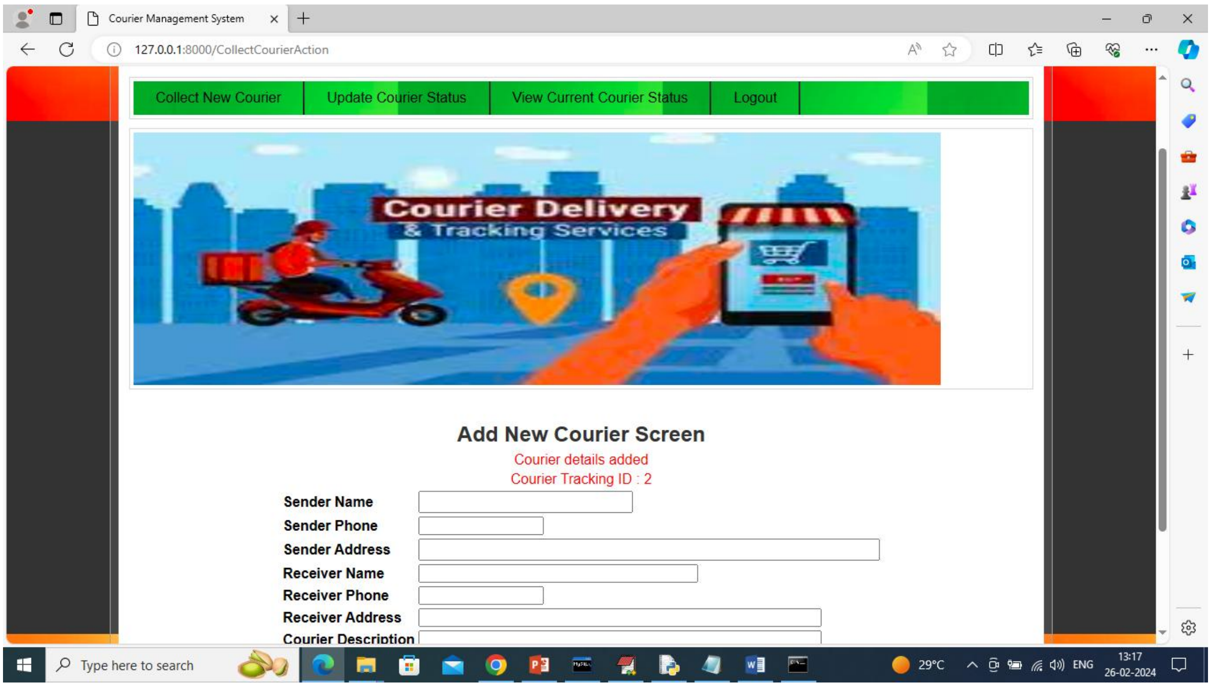


File name: 2

Add New Courier Screen

Sender Name	Ben
Sender Phone	6667778881
Sender Address	9-4-138/ 9 Mehdipatnam, Telangana
Receiver Name	dave
Receiver Phone	5556664442
Receiver Address	Flat No 103, streel 2, Chicago, USA
Courier Description	Black colour Samsung Smart Phone
Courier Weight	500 gms
Amount	2000
Expected Delivery	29-2-2024
Courier Imaget	

In above screen employee will collect all courier details and then upload courier item image and then press 'Submit' button to add new courier details and get below page


Courier Management System
127.0.0.1:8000/CollectCourierAction
Collect New Courier
Update Courier Status
View Current Courier Status
Logout
Courier Delivery & Tracking Services
Type here to search
29°C
ENG
13:17
26-02-2024

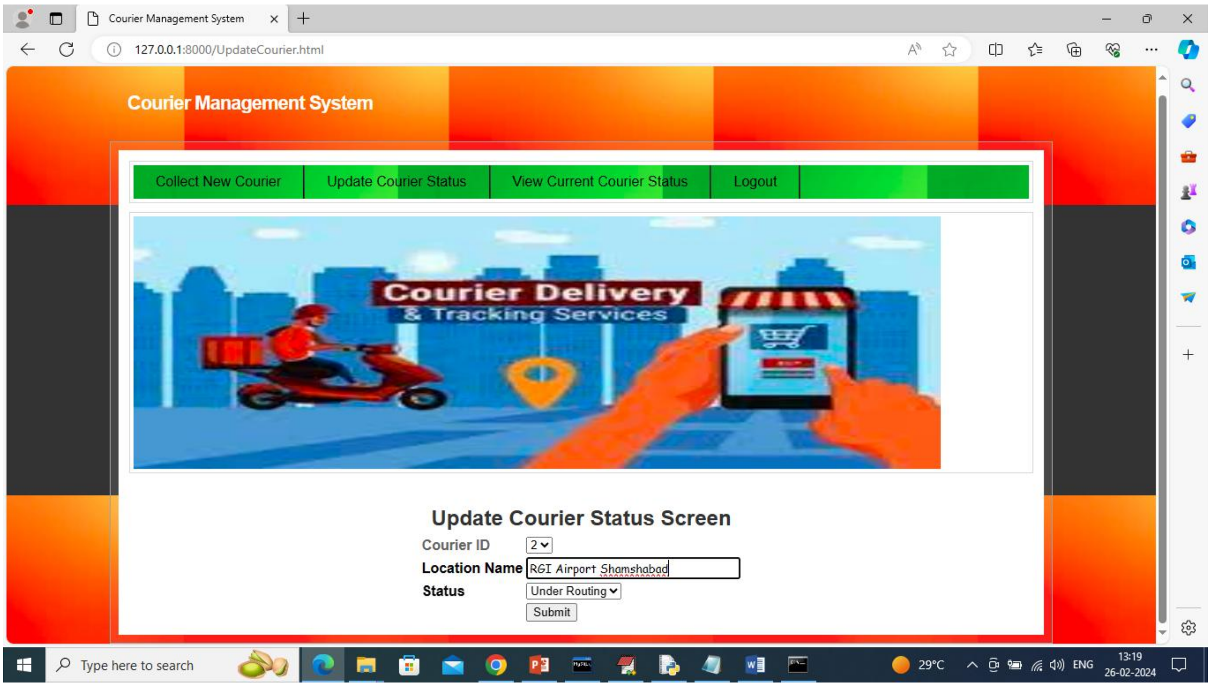
Add New Courier Screen

Courier details added

Courier Tracking ID : 2

Sender Name	
Sender Phone	
Sender Address	
Receiver Name	
Receiver Phone	
Receiver Address	
Courier Description	

In above screen courier details added and this courier can be tracked using ID as 2 and upon courier progress then employee will click on 'Update Courier Status' link to update location and get below page



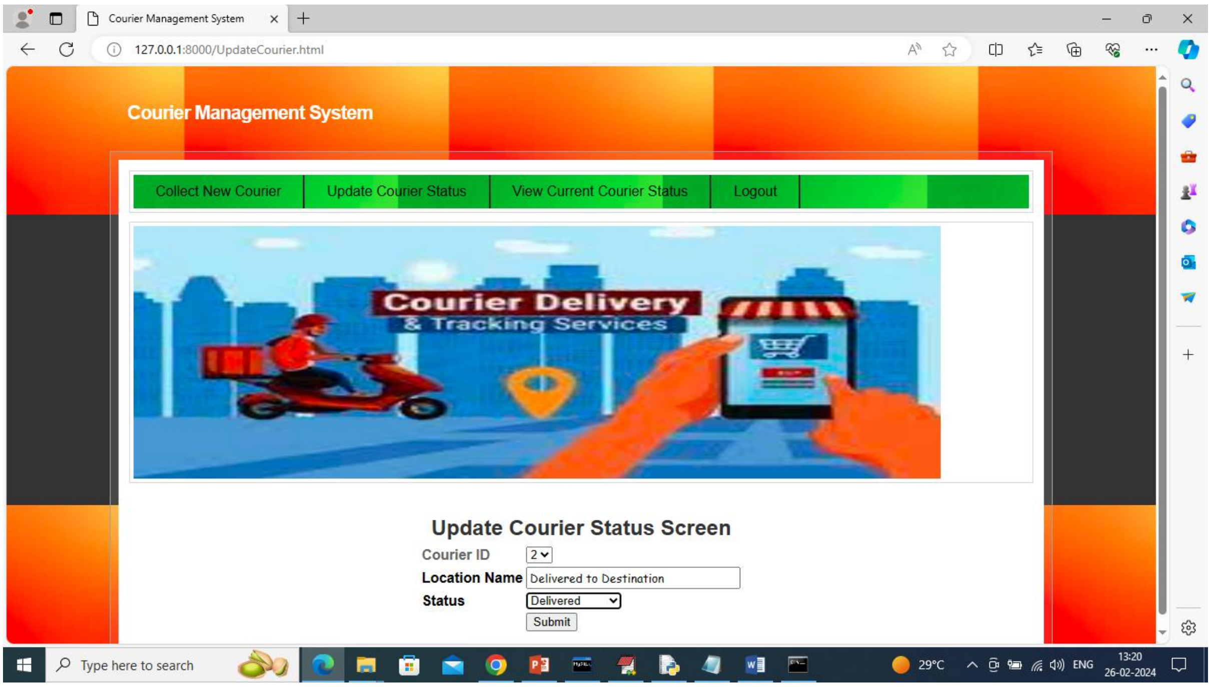
Update Courier Status Screen

Courier ID 2

Location Name RGI Airport Shamshabad

Status Under Routing

In above screen employee will select courier ID and update current location and this process continues till courier delivered and once delivered then he will update status as 'Delivered'



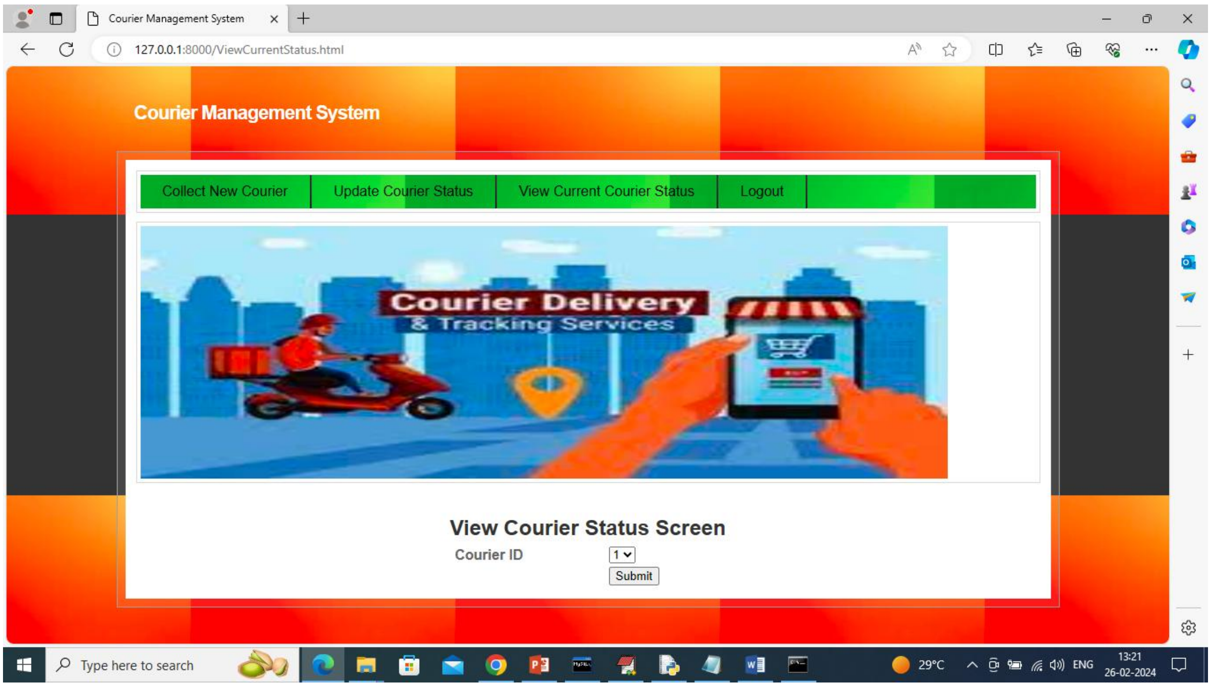
Update Courier Status Screen

Courier ID 2

Location Name Delivered to Destination

Status Delivered

In above screen one courier delivered then employee will choose status as 'Delivered' and update its status and now click on 'View Courier Current Status' link to get below page



Courier Management System

127.0.0.1:8000/ViewCurrentStatus.html

Courier Management System

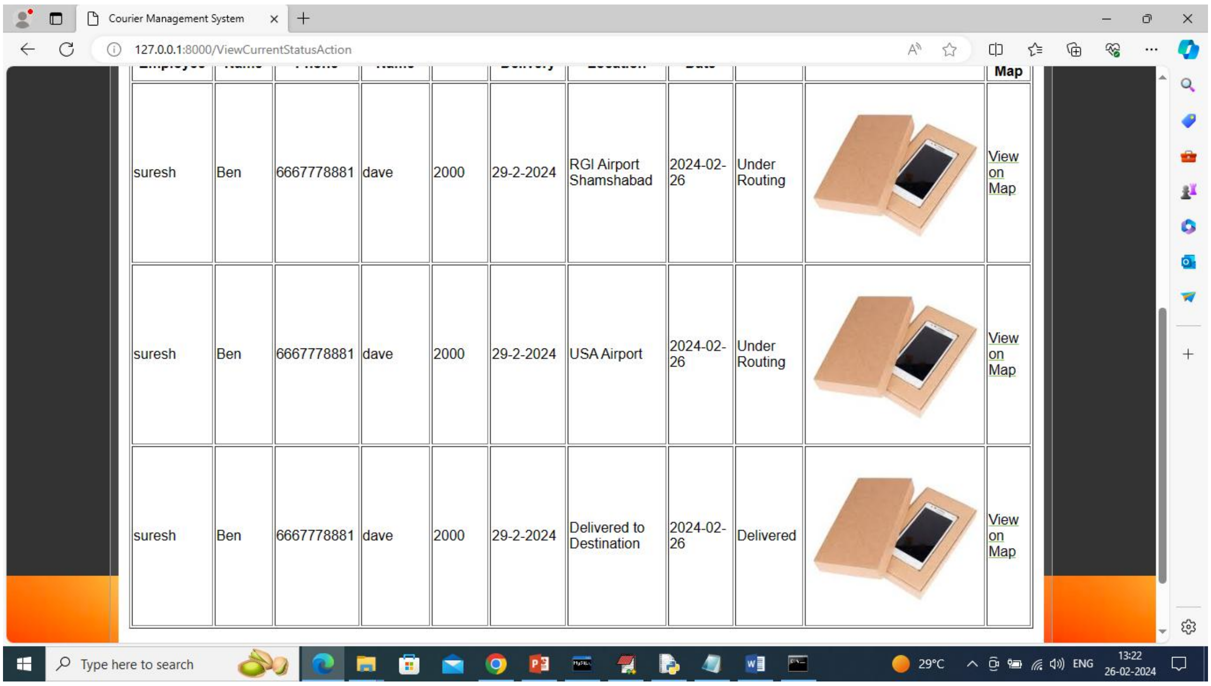
Collect New Courier | Update Courier Status | View Current Courier Status | Logout

View Courier Status Screen

Courier ID 1

Submit

In above screen select courier ID and then click on ‘Submit’ button to get below page

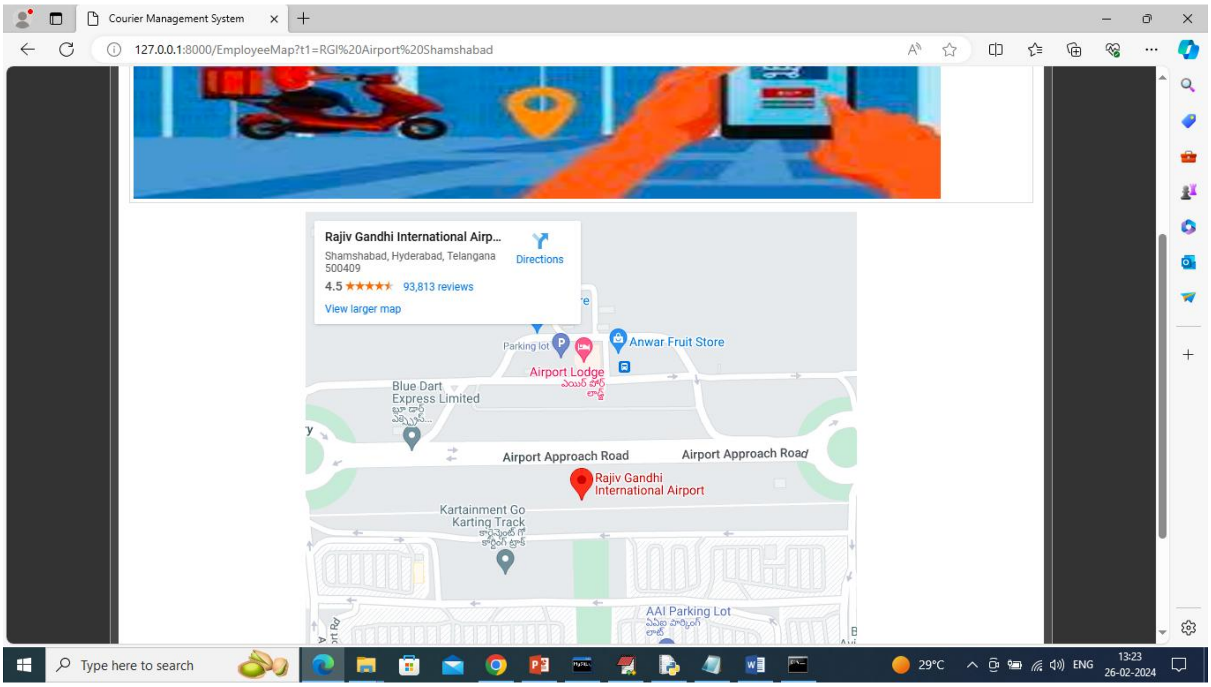


Courier Management System

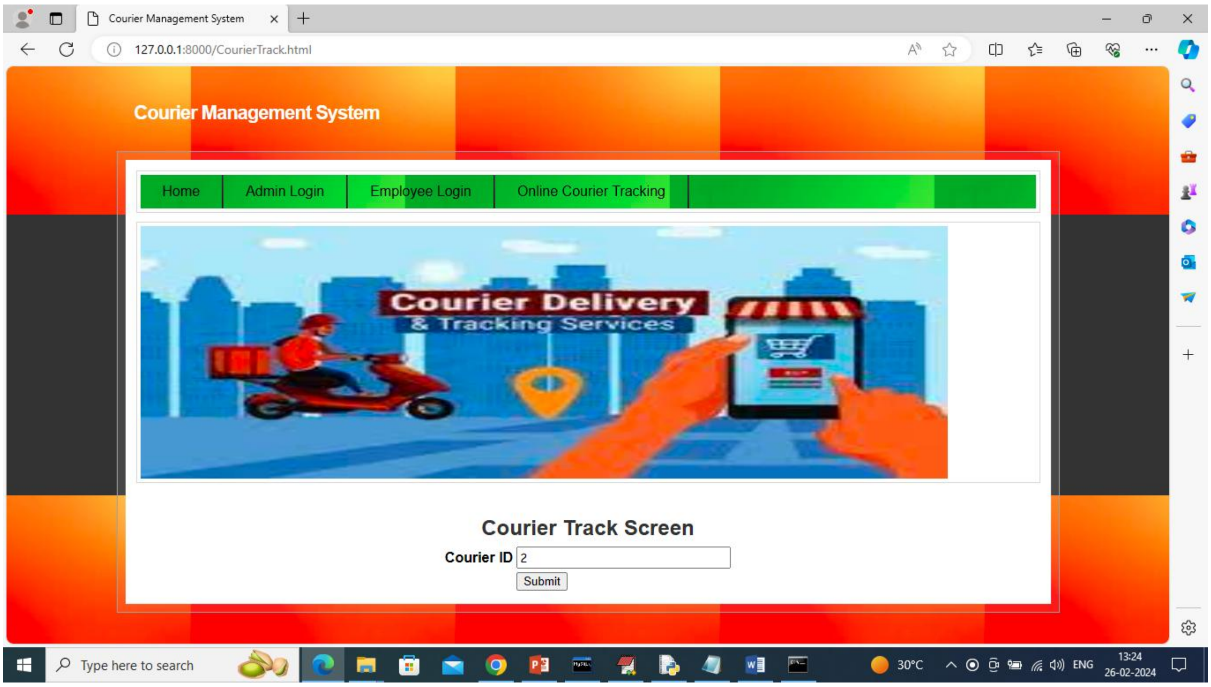
127.0.0.1:8000/ViewCurrentStatusAction

										Map
suresh	Ben	6667778881	dave	2000	29-2-2024	RGI Airport Shamshabad	2024-02-26	Under Routing		View on Map
suresh	Ben	6667778881	dave	2000	29-2-2024	USA Airport	2024-02-26	Under Routing		View on Map
suresh	Ben	6667778881	dave	2000	29-2-2024	Delivered to Destination	2024-02-26	Delivered		View on Map

In above screen for selected courier ID employee can see current location and once delivered then status will be shown as ‘Delivered’ and can click on ‘View on Map’ link to get below page


Courier Management System
127.0.0.1:8000/EmployeeMap?t1=RGI%20Airport%20Shamshabad
Rajiv Gandhi International Airp...
Shamshabad, Hyderabad, Telangana 500409
4.5 ★★★★★ 93,813 reviews
Directions
View larger map
Anwar Fruit Store
Parking lot
Airport Lodge
Blue Dart Express Limited
Airport Approach Road
Airport Approach Road
Rajiv Gandhi International Airport
Kartainment Go Karting Track
AAI Parking Lot
Type here to search
29°C
ENG
13:23
26-02-2024

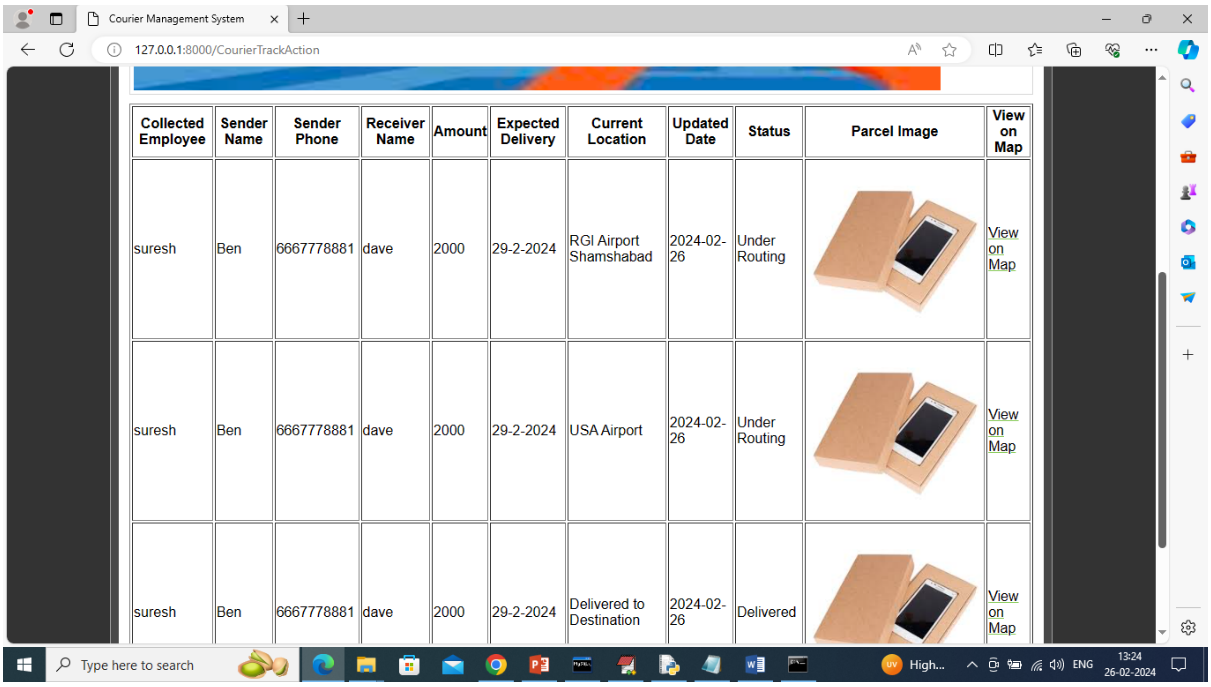
In above screen employee can see ‘courier current location was RGI Airport’. Now logout and track same with user



Courier Track Screen

Courier ID 2

In above screen user also can track courier just by entering ID and then press button to get below page



Collected Employee	Sender Name	Sender Phone	Receiver Name	Amount	Expected Delivery	Current Location	Updated Date	Status	Parcel Image	View on Map
suresh	Ben	6667778881	dave	2000	29-2-2024	RGI Airport Shamshabad	2024-02-26	Under Routing		View on Map
suresh	Ben	6667778881	dave	2000	29-2-2024	USA Airport	2024-02-26	Under Routing		View on Map
suresh	Ben	6667778881	dave	2000	29-2-2024	Delivered to Destination	2024-02-26	Delivered		View on Map

In above screen user can also see all possible location of his courier. Similarly by following above screens you can manage and run all courier system.

CONCLUSION AND FUTURE SCOPE

6.CONCLUSION AND FUTURE SCOPE

6.1 CONCLUSION

The Courier Management System developed using Python Django provides an efficient, user-friendly, and scalable solution to automate and streamline courier operations. The system successfully meets the core objectives of the project, which include efficient tracking of couriers, secure user authentication, dynamic status updates, feedback submission, and image upload capabilities.

Through this platform, administrators can manage users and gain real-time insights into all courier activities, while employees are equipped with tools to handle bookings, manage proof of delivery, and keep status updates accurate and current. Customers benefit from an intuitive interface where they can track their packages and provide feedback, enhancing overall satisfaction and transparency.

The implementation of modern software architecture and secure back-end logic ensures high system performance, security, and data consistency. Features like role-based access control, DFDs, UML modeling, and structured database design have contributed to a well-organized development process, ensuring that the system remains robust and adaptable for future expansions.

Furthermore, the integration of features such as courier image uploads and a feedback module adds a layer of reliability and credibility to the service. This makes the platform not only a management system but also a bridge for communication between service providers and customers.

In conclusion, this project demonstrates how a web-based system can effectively solve real-world logistical problems by leveraging open-source tools and following software engineering principles. With potential future enhancements—such as SMS/email notifications, GPS integration, and mobile app support—this Courier Management System can evolve into a fully-fledged logistics solution capable of serving businesses at a much larger scale.

6.2 FUTURE SCOPE

While the current implementation of the Courier Management System provides a strong foundation for managing courier operations efficiently, there is significant scope for future enhancements to expand its capabilities and improve usability, scalability, and integration with modern technologies. The following are some of the potential enhancements that can be implemented in subsequent versions of the system:

Mobile Application Integration

A dedicated mobile application for Android and iOS platforms can be developed for customers and employees. This would enhance accessibility and allow real-time tracking, courier bookings, and status updates on the go, thereby improving overall user engagement.

Real-Time GPS Tracking

Integrating GPS technology would allow for real-time tracking of courier locations. Customers would be able to see the current location of their packages, and employees could plan more efficient delivery routes, optimizing fuel usage and delivery time.

SMS and Email Notifications

To keep users informed about important updates like booking confirmations, delivery status changes, and feedback replies, an automated notification system can be added. This will improve communication and customer satisfaction.

Advanced Analytics and Reporting

Incorporating analytics tools can help the admin generate dynamic reports and dashboards. This could include performance reports, delivery trends, employee efficiency, and customer satisfaction analytics, enabling data-driven decision-making.

Barcode/QR Code Scanning

Automating package handling with barcode or QR code scanning can improve accuracy and speed in processing courier items. Employees can scan codes during pickup and delivery to instantly update the system.

AI-Based Route Optimization

An AI module can be introduced to suggest optimal delivery routes based on traffic, distance, and delivery priorities. This would help reduce fuel costs and improve delivery speed.

Digital Signature and Proof of Delivery (POD)

Enabling customers to sign digitally at the time of delivery and uploading digital PODs can further strengthen the authenticity and traceability of the delivery process.

Chatbot for Customer Support

Integrating a chatbot powered by natural language processing (NLP) could assist users with basic inquiries, track shipments, and provide support 24/7 without requiring human intervention.

Multi-Language Support

Implementing multilingual support will help cater to a more diverse user base, especially in a country like India where users prefer interacting in regional languages.

Integration with Payment Gateways

To facilitate online booking and payment, integration with popular payment gateways (such as Razorpay, Paytm, or Stripe) can be considered. This will also support the monetization of premium courier services.

These enhancements will not only future-proof the system but also make it more competitive and aligned with current industry standards. With continued development and user feedback, the Courier Management System can evolve into a comprehensive, feature-rich logistics solution for businesses of all sizes.

REFERENCES

7.REFERENCES

Bootstrap Framework – https://getbootstrap.com/docs/ **Books and Journals**

Sommerville, Ian. Software Engineering (10th Edition). Pearson Education, 2015.

Pressman, Roger S. Software Engineering: A Practitioner's Approach (8th Edition). McGraw-Hill Education, 2014.

Dennis, Alan, Barbara Haley Wixom, and Roberta M. Roth. Systems Analysis and Design. Wiley, 2015.

Research Papers

Gupta, R., & Sharma, P. (2021). Logistics Management System: A Review on Digital Transformation and Future Trends. International Journal of Advanced Research in Computer Science, 12(3), 56-61.

Kumar, S., & Raj, R. (2020). Real-Time Tracking and Route Optimization in Courier Systems using IoT and AI. International Journal of Computer Applications, 177(41), 24-29.

Web Resources

GeeksforGeeks – Tutorials and code references for backend and frontend development.

Stack Overflow – Community-based troubleshooting and discussions related to web development issues.

W3Schools – Reference for HTML, CSS, JavaScript, and PHP implementation.

GitHub – Code repositories and version control resources used for project collaboration.

Technologies & Frameworks

PHP Official Documentation – https://www.php.net/docs.php

MySQL Reference Manual – https://dev.mysql.com/doc/