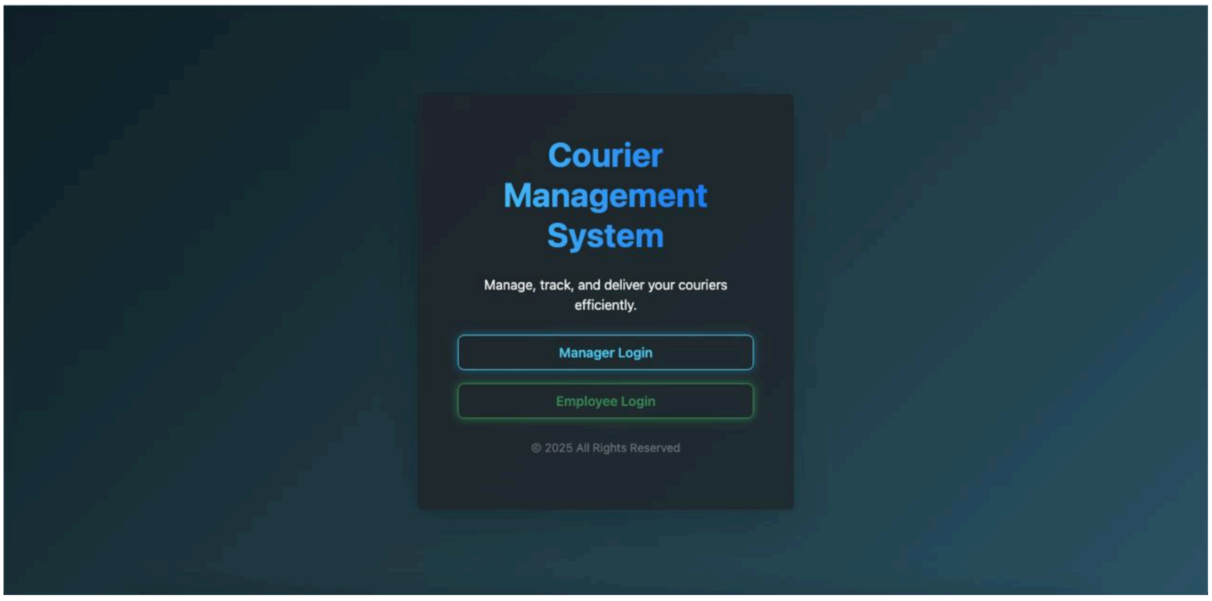
# Courier Management System

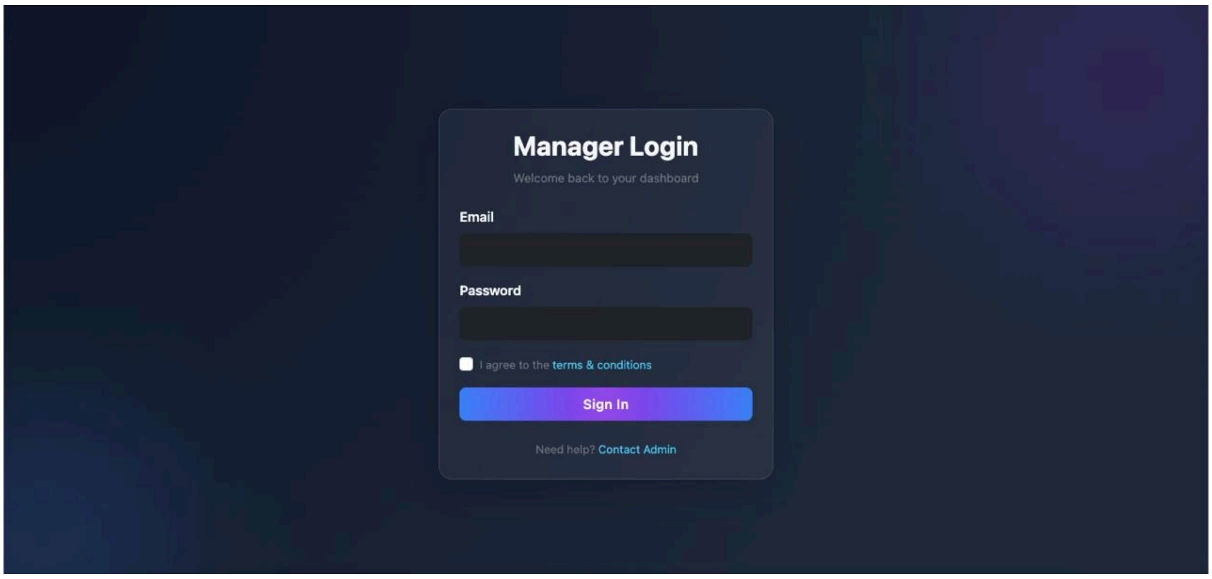
**A full-stack, AI-assisted web application for efficient courier logistics and intelligent delivery management.**

The **Courier Management System (CMS)** is a robust, end-to-end platform designed to optimize courier operations through **role-based access**, **real-time delivery tracking**, and **data-driven insights**. Built using modern web technologies, it empowers managers to oversee employees, assign deliveries, and monitor progress seamlessly — while enabling delivery personnel to update courier statuses instantly from their portal.

Developed with **Node.js, Express.js, React, and MySQL**, the CMS ensures secure, scalable, and high-performance management of courier data. Its intuitive dashboard visualizes delivery metrics and employee activity, helping organizations make informed operational decisions.

## Visual Showcase


Courier Management System
Manage, track, and deliver your couriers efficiently.
Manager Login
Employee Login
© 2025 All Rights Reserved



## Manager Login

Welcome back to your dashboard

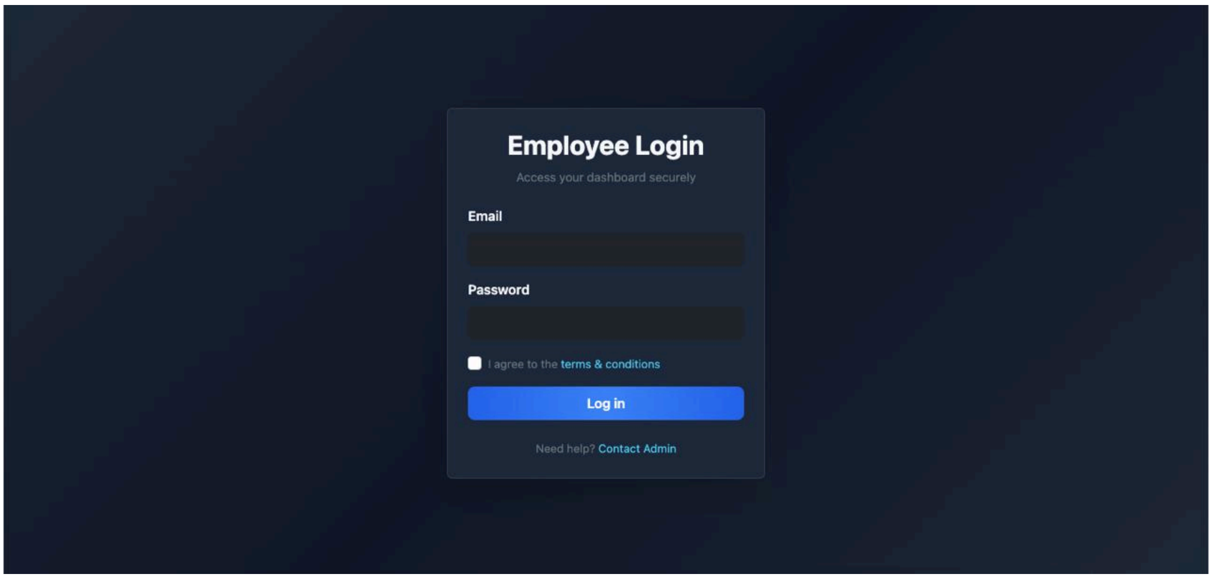
Email

Password

I agree to the terms & conditions

Sign In

Need help? Contact Admin



## Employee Login

Access your dashboard securely

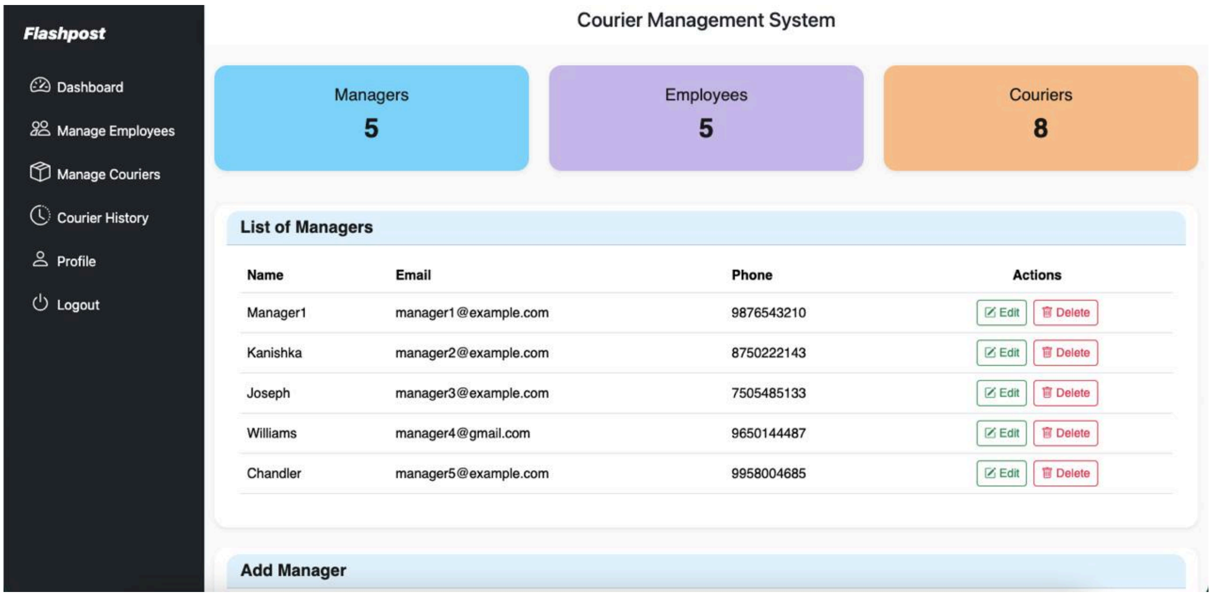
Email

Password

I agree to the terms & conditions

Log in

Need help? Contact Admin



Flashpost

- Dashboard
- Manage Employees
- Manage Couriers
- Courier History
- Profile
- Logout

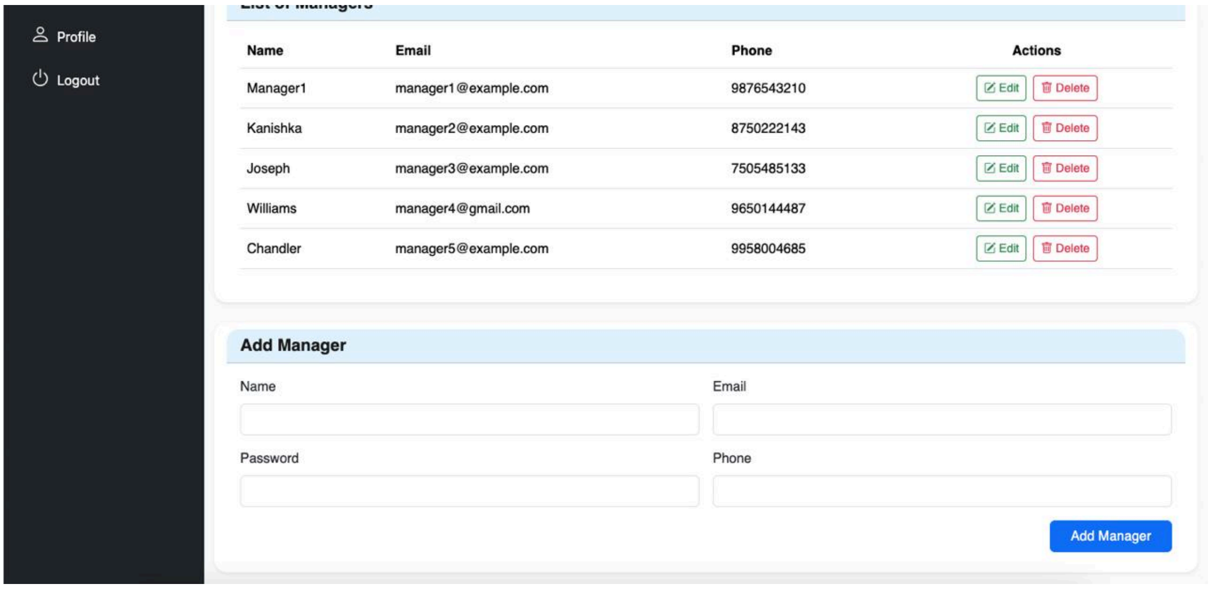
Courier Management System

| Managers | Employees | Couriers |
|---|---|---|
| 5 | 5 | 8 |

**List of Managers**

| Name | Email | Phone | Actions |
|---|---|---|---|
| Manager1 | manager1@example.com | 9876543210 | Edit Delete |
| Kanishka | manager2@example.com | 8750222143 | Edit Delete |
| Joseph | manager3@example.com | 7505485133 | Edit Delete |
| Williams | manager4@gmail.com | 9650144487 | Edit Delete |
| Chandler | manager5@example.com | 9958004685 | Edit Delete |

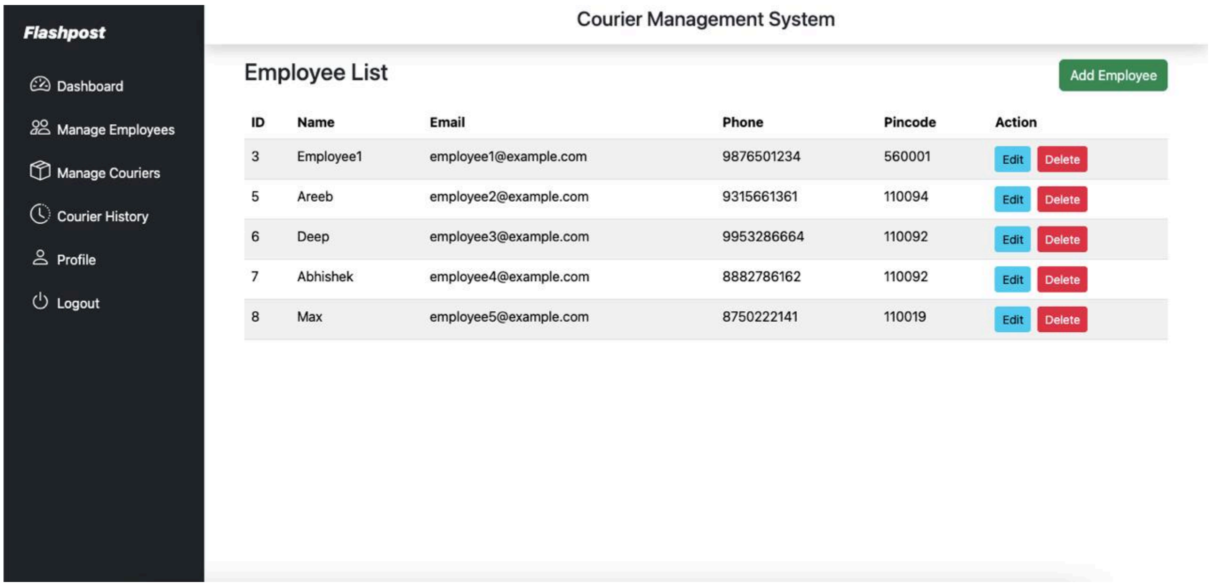
**Add Manager**



| Name | Email | Phone | Actions |
|---|---|---|---|
| Manager1 | manager1@example.com | 9876543210 | Edit Delete |
| Kanishka | manager2@example.com | 8750222143 | Edit Delete |
| Joseph | manager3@example.com | 7505485133 | Edit Delete |
| Williams | manager4@gmail.com | 9650144487 | Edit Delete |
| Chandler | manager5@example.com | 9958004685 | Edit Delete |

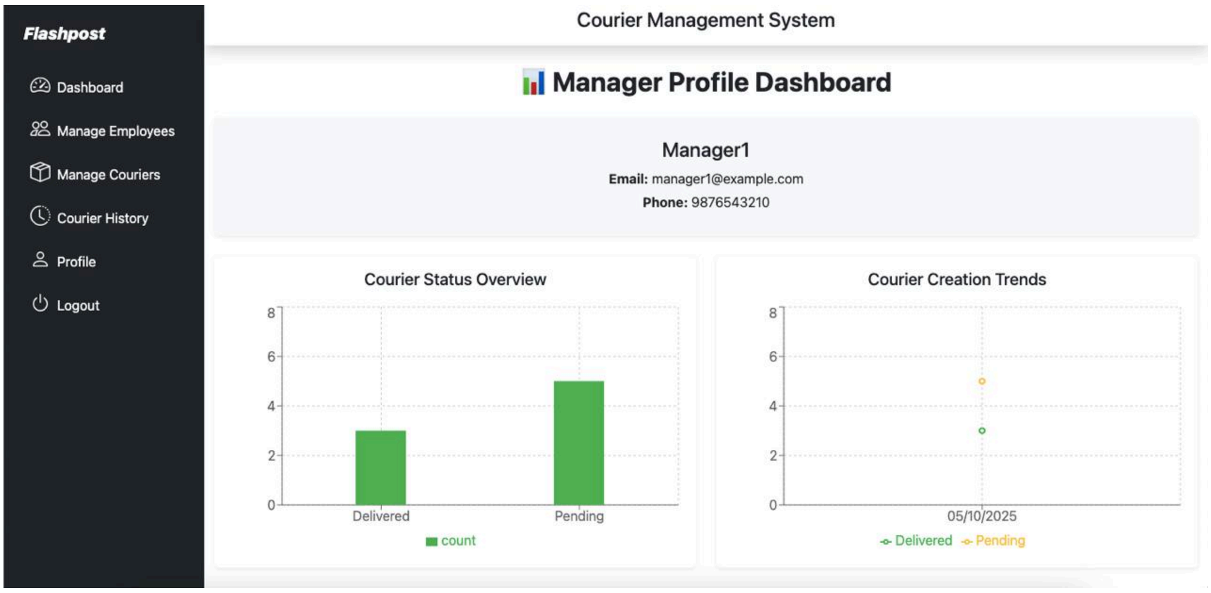
**Add Manager**

| | |
|---|---|
| Name | Email |
| Password | Phone |



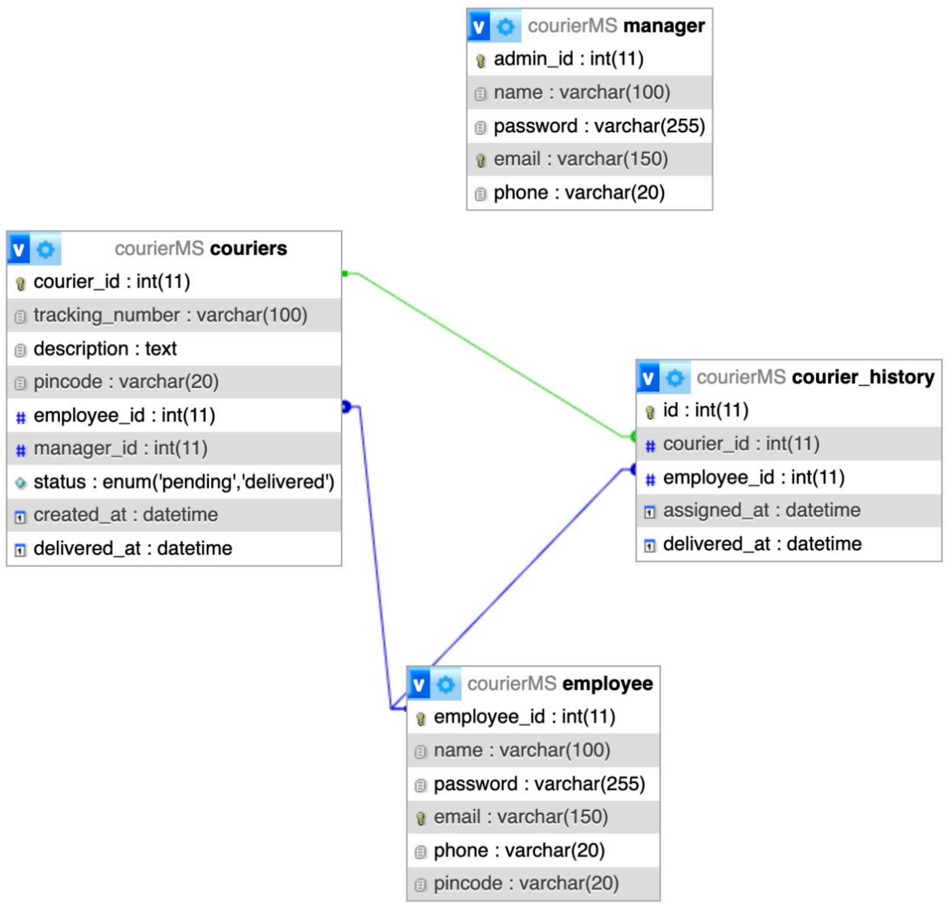
Employee List

| ID | Name | Email | Phone | Pincode | Action |
|---|---|---|---|---|---|
| 3 | Employee1 | employee1@example.com | 9876501234 | 560001 | Edit Delete |
| 5 | Areeb | employee2@example.com | 9315661361 | 110094 | Edit Delete |
| 6 | Deep | employee3@example.com | 9953286664 | 110092 | Edit Delete |
| 7 | Abhishek | employee4@example.com | 8882786162 | 110092 | Edit Delete |
| 8 | Max | employee5@example.com | 8750222141 | 110019 | Edit Delete |



Manager1
**Email:** manager1@example.com
**Phone:** 9876543210

## ER Diagram



The **Entity-Relationship (ER) Diagram** represents the core data model of the **Courier Management System**, capturing how couriers, employees, managers, and delivery records interact within the database.

1. Manager Table

- **Entity Name:** manager
- **Primary Key:** admin_id
- **Description:** Stores authentication and profile details of managers (administrators) who oversee courier operations and employee activities.
- **Key Attributes:**
  - name: Manager’s full name
  - password: Encrypted password for login authentication

    - email, phone: Contact details

Managers are responsible for adding employees, assigning couriers, and monitoring deliveries.

2. Employee Table

- **Entity Name:** employee
- **Primary Key:** employee_id
- **Description:** Contains details of delivery personnel responsible for handling courier assignments.
- **Key Attributes:**
    - name, email, phone: Employee's personal information
    - pincode: Defines the delivery zone or region the employee covers
    - password: Encrypted for secure authentication

**Relationship:**
Each employee is created and managed by a **Manager**, and can be assigned multiple **Couriers**.

3. Couriers Table

- **Entity Name:** couriers
- **Primary Key:** courier_id
- **Description:** The central entity storing all courier shipment information.
- **Key Attributes:**
    - tracking_number: Unique identifier for each courier package
    - description: Text details about the shipment contents
    - pincode: Delivery location identifier
    - status: Enum field (pending, delivered) to track courier progress
    - created_at, delivered_at: Timestamps for shipment lifecycle

**Relationships:**

- **Manager → Couriers:** One-to-Many — Each manager can create and oversee multiple courier entries (manager_id).
- **Employee → Couriers:** One-to-Many — Each courier is assigned to one employee for delivery (employee_id).

4. Courier History Table

- **Entity Name:** courier_history
- **Primary Key:** id
- **Description:** Tracks the assignment and delivery timeline for each courier. It ensures a historical record of which employee handled which courier and when.
- **Key Attributes:**

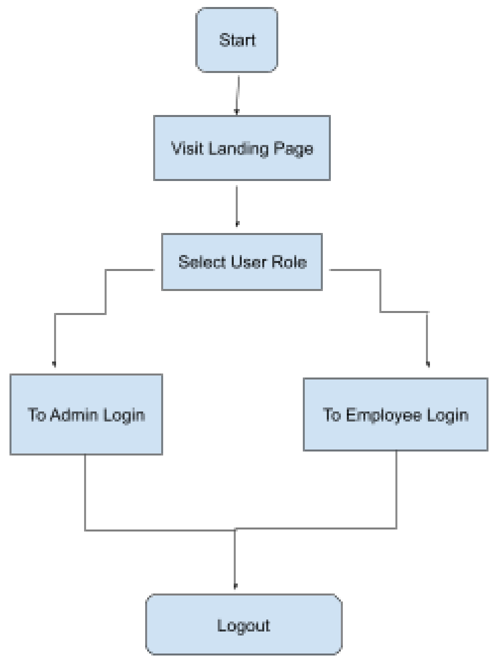
  - courier_id: References the courier being tracked
  - employee_id: References the employee responsible for the delivery
  - assigned_at: Timestamp when courier was assigned
  - delivered_at: Timestamp when delivery was completed

**Relationships:**

- **Couriers → Courier History:** One-to-Many — A courier can have multiple historical records if reassigned.
- **Employee → Courier History:** One-to-Many — An employee can handle multiple couriers over time.

## Workflow

The workflow diagrams illustrate how different user roles — Managers (Admins) and Employees (Delivery Personnel) — interact with the Courier Management System from login to logout. The flow ensures a clear separation of privileges and tasks between both roles, maintaining efficiency, data integrity, and security throughout the system.



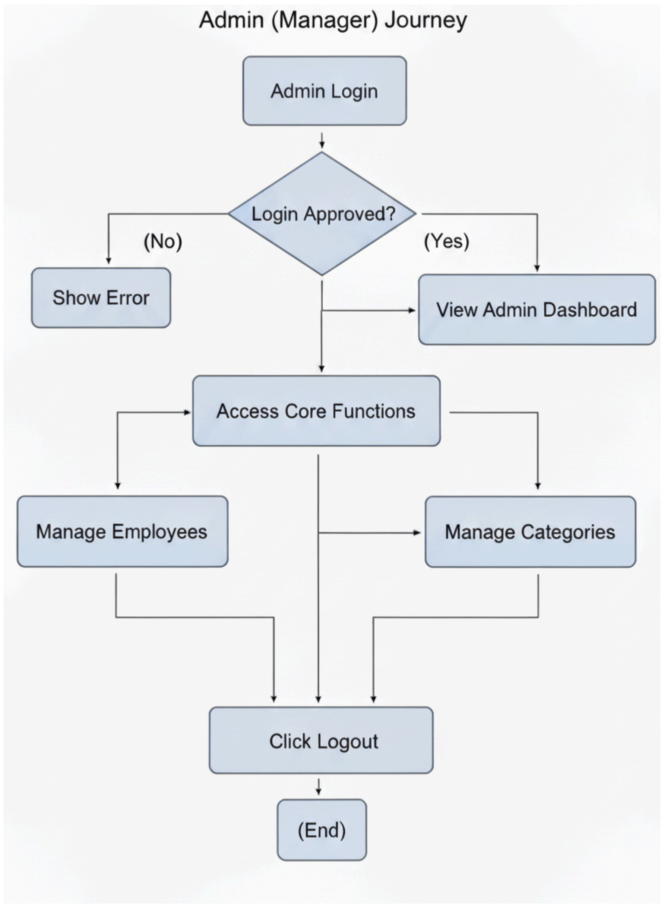
### 1. Application Start & Role Selection

When the application starts, users land on the **Home Page (Landing Page)** where they must **select their role**:

- **Admin Login** (Manager Portal)
- **Employee Login** (Delivery Personnel Portal)

Depending on the selected role, the system redirects users to the respective login page.
This ensures that each user accesses only the functionalities relevant to their responsibilities.

After completing their respective tasks, both roles have access to a **Logout option**, which securely ends their session by clearing the authentication token.



## 2. Admin (Manager) Journey

The **Manager** (or Admin) has the highest level of access and is responsible for system-wide management and analytics.

**Flow Explanation:**

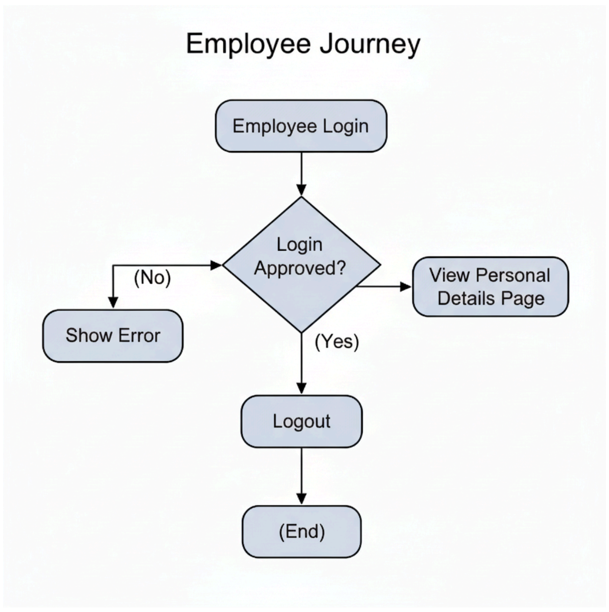
1. The admin begins by logging in via the **Admin Login Page**.
2. The system verifies credentials through the backend authentication API using **JWT tokens**.
3. If login is **not approved**, an **error message** is displayed (e.g., “Invalid credentials”).
4. If login is **approved**, the manager is redirected to the **Admin Dashboard**, which provides access to core management features.

**Core Functionalities Accessible:**

- **Manage Employees:**
  The admin can view, add, edit, or delete employee records.
- **Manage Couriers:**
  The admin can create new courier entries, assign couriers to employees, update courier details, and monitor statuses.

- **Analytics Dashboard:**
  The admin can visualize courier delivery trends and employee performance using graphical insights powered by Chart.js.

Once tasks are completed, the admin can click **Logout** to securely end the session and return to the role selection page.



## 3. Employee Journey

The **Employee** role represents delivery staff responsible for executing courier deliveries assigned by the manager.

**Flow Explanation:**

1. The employee logs in through the **Employee Login Page** using valid credentials.
2. The system verifies credentials and role-based permissions.
3. If **login fails**, an **error message** is displayed.
4. If **login is successful**, the employee is redirected to their **Personal Dashboard**, where they can:
   - View all couriers assigned to them.
   - Update courier status in real-time (e.g., “Pending” → “Delivered”).
   - Track assigned deliveries and timestamps.
5. Once the employee completes their delivery updates, they can **Logout**, securely ending their session.

## Tech Stack & Libraries

- **Frontend: React**, **React Router** (for client-side routing), **Axios** (for API requests), and **Chart.js** (for data visualization).
- **Backend: Node.js** and **Express.js** (for building the REST API).
- **Database: MySQL** (Hosted on **Railway**).
- **Authentication: JSON Web Tokens (JWT)** (for secure, stateless user sessions).
- **Styling & UI: Bootstrap**, **React-Toastify** (for user notifications), and custom **CSS**.

## Key Features

### General

- **Secure Authentication:** Role-based login system for both Managers and Employees.

### Manager Portal

- **Analytics Dashboard:** Visualizes the number of delivered couriers over time using line graphs and histograms.
- **Courier Management:** Full CRUD (Create, Read, Update, Delete) functionality for all couriers in the system.
- **Employee Management:** View, add, and manage employee details.
- **Efficient Assignment:** Assign couriers to employees with an intelligent search feature based on delivery pincode.
- **Delivery History:** Access a comprehensive history of all past and present courier statuses.

### Employee Portal

- **Assigned Deliveries:** View a clear list of all couriers assigned for delivery.
- **Status Updates:** Update the status of a courier in real-time (e.g.,"Delivered," "pending").

### API Endpoints

The backend exposes the following REST API endpoints. The base URL is /auth.

| Method | Endpoint | Description |
|---|---|---|
| POST | /adminlogin | Authenticates a manager and returns a JWT. |
| GET | /logout | Logs out the user by clearing the authentication cookie. |
| POST | /add_employee | Creates a new employee record (Manager). |
| GET | /employee | Retrieves a list of all employees (Manager). |

| | | |
|---|---|---|
| GET | /employee/:id | Retrieves a single employee by their ID (Manager). |
| PUT | /edit_employee/:id | Updates an existing employee's details (Manager). |
| DELETE | /delete_employee/:id | Deletes an employee record (Manager). |
| GET | /admin_count | Gets the total count of managers. |
| GET | /employee_count | Gets the total count of employees. |
| GET | /admin_records | Retrieves a list of all managers. |

## Design and Architectural Decisions

- **Database Choice**: I chose **MySQL** because of its reliability and the relational nature of the project's data. An employee management system has clear relationships between entities like employees, administrative roles, and categories, making a structured SQL database a natural fit for ensuring data integrity and consistency.
- **Backend Architecture**: The backend is built with **Node.js and Express.js**, creating a lightweight and efficient REST API. For handling image uploads, **Multer** was integrated as middleware to process form-data, save files to the server's filesystem, and store the file path in the database.
- **Authentication**: I implemented **JSON Web Tokens (JWT)** for secure, stateless authentication. After a successful login, a JWT is generated on the server and stored in a browser cookie. This token is then sent with subsequent requests to verify the user's session and role without the server needing to store session state.
- **State Management**: The frontend relies on **React Hooks** (useState, useEffect) for managing component-level state. For a CRUD application of this scope, this approach keeps the state management logic simple and localized, avoiding the boilerplate and complexity of a global state management library like Redux or the Context API.

### Trade-offs and Future Improvements

- **Trade-off - UI/UX**: The primary focus was on implementing the core backend and frontend functionality. I used the **Bootstrap** library for rapid UI development, which meant prioritizing functionality over a highly customized or unique design.

- **Future Improvement - Role-Based Access Control (RBAC)**: The video series lays the groundwork for this. The next logical step is to build out the employee-facing side of the application. This would involve creating a separate login and dashboard for employees, allowing them to view their own details and perform limited actions, while restricting administrative functions to manager/admin roles.
- **Future Improvement - Enhanced Password Security**: While passwords are securely hashed using **bcrypt**, the system could be improved by adding a "forgot password" feature that sends a secure reset link to the user's email.
- **Future Improvement - Advanced Dashboard Analytics**: The current dashboard shows basic counts. This could be expanded with more detailed visualizations, such as charts for salary distribution across categories or employee tenure, providing more meaningful insights for management.

## AI Integration

**1.** Prompt on Debugging and Database Linking

“My ManageCouriers.jsx page is not displaying pending and delivered couriers, even though the data exists in my MySQL database. Help me identify why the data isn’t rendering and fix the connection between the frontend, backend, and database without changing the overall UI or functionality.”

2.Prompt on UI Enhancement

“Change the UI for EmployeeLogin.jsx to make it look more professional and responsive, but keep the same color scheme and minimal aesthetic. It should still retain all its functionality, including validation and navigation after login.”

3. Prompt on Feature Integration

“I want the EmployeeDetail page to look like a dashboard where the employee can view all assigned couriers, see their own details like name, email, pincode, and phone number, and also mark couriers as delivered. These updates should reflect automatically on the manager’s side.”

4. Prompt on Report Generation

“Add a feature that allows the manager to generate a report showing the number of delivered couriers in a specific pincode within a selected date range. The results should be clearly displayed below the form.”

5. Prompt on Deployment Assistance

“I want to deploy this full-stack project with a Node.js backend, MySQL database, and React frontend. Tell me the easiest and most reliable way to do it, and explain step-by-step what files or configurations I should include.”

6. Prompt on Data Visualization

“Update the Profile.jsx page to display a meaningful graph that visualizes courier delivery data based on what is being collected. The graph should be easy to read and fit naturally into the existing dashboard.”