

A CAB WESITE FOR A COMPANY



A PROJECT REPORT

Submitted by

VAITHYANADHAN S G (2303811724321119)

in partial fulfillment of requirements for the award of the course

CGB1221-DATABASE MANAGEMENT SYSTEMS

in

ARTIFICIAL INTELLIGENCE AND DATA SCIENCE

K. RAMAKRISHNAN COLLEGE OF TECHNOLOGY

(An Autonomous Institution, affiliated to Anna University Chennai and Approved by AICTE, New Delhi)

SAMAYAPURAM – 621 112

JUNE - 2025

**K. RAMAKRISHNAN COLLEGE OF TECHNOLOGY
(AUTONOMOUS)**

SAMAYAPURAM – 621 112

BONAFIDE CERTIFICATE

Certified that this project report on “ **A Cab Website For a Company**” is the bonafide work of **VAITHYANADHAN S G (2303811724321119)** who carried out the project work during the academic year 2024 - 2025 under my supervision.

SIGNATURE

Dr.T. AVUDAIAPPAN, M.E.,Ph.D.,

HEAD OF THE DEPARTMENT

ASSOCIATE PROFESSOR

Department of Artificial Intelligence

K.Ramakrishnan College of Technology
(Autonomous)

Samayapuram–621112.

SIGNATURE

Mrs.S. GEETHA, M.E.,

SUPERVISOR

ASSISTANT PROFESSOR

Department of Artificial Intelligence

K.Ramakrishnan College of Technology
(Autonomous)

Samayapuram–621112.

Submitted for the viva-voce examination held on..... 04.06.2025

INTERNAL EXAMINER

EXTERNAL EXAMINER

DECLARATION

I declare that the project report on “**A Cab website for a Company**” is the result of original work done by me and best of my knowledge, similar work has not been submitted to “**ANNA UNIVERSITY CHENNAI**” for the requirement of Degree of **BACHELOR OF TECHNOLOGY**. This project report is submitted on the partial fulfilment of the requirement of the completion of the course **CGB1221 – DATABASE MANAGEMENT SYSTEMS**.

Signature

VAITHYANADHAN S G

Place: Samayapuram

Date: 04.06.2025

ACKNOWLEDGEMENT

It is with great pride that I express our gratitude and in-debt to our institution “**K.Ramakrishnan College of Technology (Autonomous)**”, for providing us with the opportunity to do this project.

I glad to credit honourable chairman **Dr. K. RAMAKRISHNAN, B.E.**, for having provided for the facilities during the course of our study in college.

I would like to express our sincere thanks to our beloved Executive Director **Dr. S. KUPPUSAMY, MBA, Ph.D.**, for forwarding to our project and offering adequate duration in completing our project.

I would like to thank **Dr. N. VASUDEVAN, M.Tech., Ph.D.**, Principal, who gave opportunity to frame the project the full satisfaction.

I whole heartily thanks to **Dr. T. AVUDAIAPPAN, M.E.,Ph.D.**, Head of the department, **ARTIFICIAL INTELLIGENCE** for providing his encourage pursuing this project.

I express our deep expression and sincere gratitude to our project supervisor **Mrs.S.GEETHA, M.E.**, Department of **ARTIFICIAL INTELLIGENCE**, for her incalculable suggestions, creativity, assistance and patience which motivated us to carry out this project.

I render our sincere thanks to Course Coordinator and other staff members for providing valuable information during the course.

I wish to express our special thanks to the officials and Lab Technicians of our departments who rendered their help during the period of the work progress.

INSTITUTE

Vision:

- To serve the society by offering top-notch technical education on par with global standards.

Mission:

- Be a center of excellence for technical education in emerging technologies by exceeding the needs of industry and society.
- Be an institute with world class research facilities.
- Be an institute nurturing talent and enhancing competency of students to transform them as all – round personalities respecting moral and ethical values.

DEPARTMENT

Vision:

- To excel in education, innovation, and research in Artificial Intelligence and Data Science to fulfil industrial demands and societal expectations.

Mission

- To educate future engineers with solid fundamentals, continually improving teaching methods using modern tools.
- To collaborate with industry and offer top-notch facilities in a conducive learning environment.
- To foster skilled engineers and ethical innovation in AI and Data Science for global recognition and impactful research.
- To tackle the societal challenge of producing capable professionals by instilling employability skills and human values.

PROGRAM EDUCATIONAL OBJECTIVES (PEO)

- **PEO1:** Compete on a global scale for a professional career in Artificial Intelligence and Data Science.
- **PEO2:** Provide industry-specific solutions for the society with effective communication and ethics.
- **PEO3** Enhance their professional skills through research and lifelong learning initiatives.

PROGRAM SPECIFIC OUTCOMES (PSOs)

- **PSO1:** Capable of finding the important factors in large datasets, simplify the data, and improve predictive model accuracy.
- **PSO2:** Capable of analyzing and providing a solution to a given real-world problem by designing an effective program.

PROGRAM OUTCOMES (POs)

Engineering students will be able to:

1. **Engineering knowledge:** Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
2. **Problem analysis:** Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences
3. **Design/development of solutions:** Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations
4. **Conduct investigations of complex problems:** Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions
5. **Modern tool usage:** Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations
6. **The engineer and society:** Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice
7. **Environment and sustainability:** Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development
8. **Ethics:** Apply ethical principles and commit to professional ethics and

responsibilities and norms of the engineering practice.

- 9. Individual and team work:** Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
- 10. Communication:** Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
- 11. Project management and finance:** Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
- 12. Life-long learning:** Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

ABSTRACT

The Cab Website for a Company helps company employees easily book cabs to their workplace. Users can sign up, log in, book rides, track where the cab is, see their ride history, and manage payments through a wallet. Drivers can log in to see which users need a ride, mark rides as completed, and track their earnings. The system is made using Python (Flask) for the backend, MySQL for storing data, and HTML, CSS, and JavaScript for the website design. It also has features like refunds for ride cancellations and assigning drivers based on pickup location. This project makes booking a cab simple, fast, and clear for both users and drivers.

ABSTRACT WITH POs AND PSOs MAPPING

CO 5 : BUILD DATABASES FOR SOLVING REAL-TIME PROBLEMS.

ABSTRACT	POs MAPPED	PSOs MAPPED
The Cab Website for a Company helps company employees easily book cabs to their workplace. Users can sign up, log in, book rides, track where the cab is, see their ride history, and manage payments through a wallet. Drivers can log in to see which users need a ride, mark rides as completed, and track their earnings. The system is made using Python (Flask) for the backend, MySQL for storing data, and HTML, CSS, and JavaScript for the website design. It also has features like refunds for ride cancellations and assigning drivers based on pickup location. This project makes booking a cab simple, fast, and clear for both users and drivers.	PO1 -3 PO2 -3 PO3 -3 PO4 -2 PO5 -3 PO6 -2 PO7 -1 PO8 -2 PO9 -3 PO10 -3 PO11 -2 PO12 -3	PSO1 -3 PSO2 -3

Note: 1- Low, 2-Medium, 3- High

TABLE OF CONTENTS

CHAPTER NO.	TITLE	PAGE NO.
	ABSTRACT	viii
1	INTRODUCTION	1
	1.1 OBJECTIVE	1
	1.2 OVERVIEW	2
	1.3 SQL AND DATABASE CONCEPTS	3
2	PROJECT METHODOLOGY	5
	2.1 PROPOSED WORK	5
	2.2 BLOCK DIAGRAM	6
3	MODULE DESCRIPTION	7
	3.1 USER MODULE	7
	3.2 DRIVER MODULE	7
	3.3 ADMIN MODULE	8
	3.4 REGISTER MODULE	8
	3.5 HISTORY & CALCULATION MODULE	9
4	CONCLUSION & FUTURE SCOPE	10
	APPENDIX A SOURCE CODE	11
	APPENDIX B SCREENSHOTS	20
	REFERENCES	24

CHAPTER 1

INTRODUCTION

INTRODUCTION

The Cab Booking System is a Python-based web application developed to simplify the process of booking cabs for company employees. It provides an easy-to-use interface for both users and drivers. Users can register, log in, book a cab, track the driver's location, view ride history, and manage payments through an in-built wallet system. Drivers can log in, view available ride requests, accept bookings, and update ride status. Built using Python (Flask) for the backend, MySQL for data storage, and HTML, CSS, and JavaScript for the frontend, the system offers a smooth and interactive experience. The platform improves the cab management process by enabling efficient communication between users and drivers, making ride booking and tracking simple, transparent, and organized.

1.1 OBJECTIVE

The primary objective of the Cab Booking System is to build an efficient, user-friendly platform that simplifies and streamlines the process of booking and managing cab rides for employees. The system aims to:

- 1. Automate ride booking:** Reducing manual coordination by allowing users to easily book rides and enabling drivers to manage ride assignments efficiently.
- 2. Enhance user convenience:** Providing a clear and interactive interface for booking cabs, tracking drivers, viewing ride history, and managing payments.
- 3. Demonstrate the power of Python and web technologies:** Utilizing Flask for the backend, MySQL for data handling, and HTML/CSS/JavaScript for an interactive and modern frontend experience.

- 4. Replace traditional booking methods:** Offering a digital alternative to manual cab scheduling for improved efficiency and reliability.
- 5. Improve ride management:** Enabling drivers to view ride requests, track their completed trips, and monitor earnings, while allowing users to manage their bookings and wallet.
- 6. Ensure data security:** Storing user and driver information securely using MySQL, ensuring safe transactions and privacy.

1.2 OVERVIEW

The Cab Booking System is a web-based application developed to simplify and manage employee transportation within an organization. The system provides outlined below.

- 1.** The Cab Booking System is a full-stack web application designed to simplify the process of booking rides for users while efficiently managing driver assignments and ride tracking.
- 2.** Users can register, log in, book rides, view available drivers, and track their ride history.
- 3.** Drivers have a dedicated dashboard where they can log in, set their availability, view user ride requests, accept bookings, and update ride status.
- 4.** Admins oversee the system by managing users, drivers, bookings, ride history, and financial transactions.
- 5.** The system is built using Flask (Python) for backend logic, MySQL for data storage, and HTML/CSS/JavaScript for a responsive frontend interface.
- 6.** Includes a secure wallet system: users are charged for bookings and refunded based on cancellation policy (full or 50% depending on ride

status)..

7. Drivers earn ₹100 per completed ride and ₹50 for canceled rides, which is reflected in a tabular earnings report on their dashboard.

8. The application enforces a dual-confirmation model—both user and driver must mark the ride as "completed" before a driver can be reassigned.

1.3 SQL AND DATABASE CONCEPT

1. Relational Database: The project uses MySQL, a relational database system, to store data in structured tables such as users, drivers, rides, and transactions. This allows for organized storage and easy retrieval of interconnected data using SQL queries.

2. Primary Keys: Each table includes a unique identifier called a primary key (e.g., `user_id` in the users table, `ride_id` in the rides table). This ensures that every record is uniquely identifiable and helps maintain data integrity.

3. Foreign Keys: Foreign keys are used to link data between tables, such as assigning a ride to a specific user and driver using `user_id` and `driver_id`. These relationships enable the system to join data across tables and maintain referential integrity.

4. CRUD Operations (Create, Read, Update, Delete): The system supports all basic database operations: creating new user accounts, reading booking info, updating ride status, and deleting cancelled rides. These operations are performed through SQL statements in the backend to manage dynamic user and ride data.

5. Data Normalization: The database is normalized to reduce redundancy by splitting information into multiple related tables (e.g., rides and payments). This improves data consistency, prevents duplication, and simplifies future updates or changes.

6. Transactions: All monetary exchanges such as booking charges, refunds, and wallet recharges are stored in a transactions table. This helps track financial activities, ensuring accountability and allowing audit of earnings and user balances.

7. Aggregate Functions: Functions like SUM() are used to calculate totals, such as a driver's total earnings or a user's total spending. These calculations provide summarized insights, displayed in dashboards or reports.

8. Joins: SQL JOIN statements are used to combine data from multiple tables—for example, showing driver name, user info, and ride details in one view. This is essential for presenting meaningful data on dashboards and for managing bookings.

CHAPTER 2

PROJECT METHODOLOGY

2.1 PROPOSED WORK

The project aims to build a user-friendly and efficient Cab Booking Website using web technologies such as Python (Flask), MySQL, HTML, CSS, and JavaScript. The goal is to create a platform where users (employees) can easily book cabs and drivers can manage their rides effectively. The methodology followed in developing this project includes the following steps:

- 1. Understanding Requirements:** Identifying the needs of users, such as registration, login, booking rides, viewing driver availability, and tracking ride history. Understanding driver requirements like login, viewing ride requests, updating ride status, and viewing earnings. Analyzing existing cab systems to improve user experience and remove manual booking methods.
- 2. Data storage:** Designing modular and organized code using Python and Flask to manage backend logic. Using MySQL to store user, driver, ride, and payment information securely. Structuring the database to ensure data consistency, scalability, and quick access.
- 3. Implementation:** Developing a responsive and interactive frontend using HTML, CSS, and JavaScript. Creating separate modules for users and drivers, including dashboards, ride assignment, ride completion, and wallet management. Integrating backend functionality with MySQL to handle user data, ride requests, driver earnings, and booking history.
- 4. Testing and Validation:** Testing each module individually (user login, ride booking, driver updates, etc.). Verifying integration between frontend, backend, and database. Fixing any errors and ensuring the website runs smoothly across all user actions.
- 5. Deployment and Feedback:** Deploying the completed cab booking

system for end users and drivers. Collecting feedback to improve user experience and add future enhancements like real-time maps or notifications.

2.2 BLOCK DIAGRAM

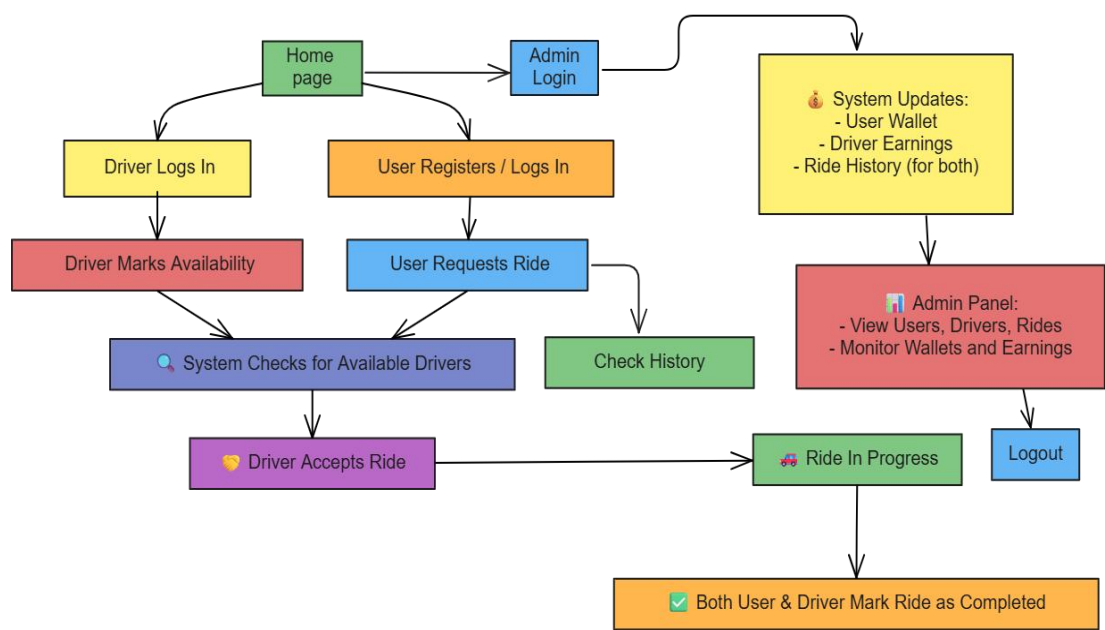


Fig.No: 2.1

CHAPTER 3

MODULE DESCRIPTION

3.1 USER MODULE

To enable users to interact with the cab booking system seamlessly-allowing them to register, log in, book rides, cancel rides with appropriate refunds, and review ride history for accountability and personal tracking.

1. **User Registration & Login:** Enables users to sign up and securely log in.
2. **User Dashboard:** Displays wallet balance, ride options, and booking status.
3. **Ride Booking:** Lets users select pickup location and request a cab.
4. **Ride Cancellation with Refund:** Allows full or 50% refund depending on ride status.
5. **Wallet System:** Manages user funds for bookings and refunds.
6. **Driver Availability View:** Users can check which drivers are currently available.

3.2 DRIVER MODULE

To let drivers manage ride assignments, update ride status, and track their earnings.

1. **Driver Login:** Allows predefined drivers to securely login.
2. **Driver Dashboard:** Displays assigned rides and earnings summary.
3. **Mark as Available:** Sets driver status as available for booking.
4. **Accept Ride:** Allows accepting new ride requests.
5. **Ride Completion:** Lets drivers mark ride as completed.

- 6. Earnings Tracker:** Calculates ₹100/completed, ₹50/cancelled ride
- 7. Ride History:** Lists past rides with status and earnings.

3.3 ADMIN MODULE

To control and monitor all users Login, New Users, drivers, rides, manages the user to pick up and drop in the company with respective driver near by the pickup station and wallet transactions across the platform

- 1. Admin Login:** Provides secure login for administrators.
- 2. Admin Dashboard:** Overview of all system data and activity.
- 3. User & Driver Management:** View, update, or reset profiles and credentials.
- 4. Ride Monitoring:** Tracks live and completed rides with user/driver info.
- 5. Refund and Cancellation Control:** Ensures rules are followed during cancellations.

3.4 REGISTER MODULE

To facilitate the registration of new users and assign pre-registered drivers securely.

- 1. User Registration:** Collects user details and creates a new account.
- 2. Driver Setup:** Drivers are registered via the database and validated at login.
- 3. Input Validation:** Ensures proper data entry and prevents duplication.

3.5 HISTORY & CALCULATION MODULE

To calculate a fair fee for each ride for the user respectively the number of ride and selecting the place of departure.

- 1. To track ride:** completed/canceled rides, user refund eligibility, and driver earnings.
- 2. User Ride History:** Shows list of past rides with fare and refund status.
- 3. Driver Ride History:** Lists all accepted rides, status, and earnings.
- 4. Refund Logic:** Applies full or 50% refund based on driver assignment.

CHAPTER 4

CONCLUSION & FUTURE SCOPE

CONCLUSION:

The Cab Booking System successfully provides a digital platform for users and drivers to manage cab rides efficiently and securely. Users can book rides, check driver availability, track their ride history, and manage their wallet with refund features based on ride status. Drivers can view available ride requests, select users based on profiles, complete rides, and monitor their earnings. The system ensures seamless interaction between users, drivers, and the admin through an intuitive interface, real-time updates, and secure database integration. Overall, it enhances the traditional cab booking experience by automating core processes, reducing manual work, and improving reliability and transparency for all parties involved.

FUTURE SCOPE:

- 1. AI-based Matching:** Implementing smart driver-user matching using location, availability, and preferences to optimize ride allocation.
- 2. Online Payment Integration:** Introducing third-party payment gateways like Razorpay, Paytm, or UPI for faster transactions.
- 3. Notification System:** Implementing SMS or email alerts for ride status updates, driver assignment, and payment confirmations.

APPENDIX A – SOURCE CODE

MySQL:

```
CREATE DATABASE IF NOT EXISTS cab_system;  
USE cab_system;
```

```
CREATE TABLE users (  
  id INT AUTO_INCREMENT PRIMARY KEY,  
  name VARCHAR(100),  
  email VARCHAR(100) UNIQUE,  
  password VARCHAR(100),  
  phone VARCHAR(20)  
);
```

```
CREATE TABLE drivers (  
  id INT AUTO_INCREMENT PRIMARY KEY,  
  name VARCHAR(100),  
  phone VARCHAR(20) UNIQUE,  
  location VARCHAR(100),  
  is_available BOOLEAN DEFAULT FALSE  
);
```

```
CREATE TABLE rides (  
  id INT AUTO_INCREMENT PRIMARY KEY,  
  user_id INT,  
  driver_id INT,  
  pickup VARCHAR(100),  
  time DATETIME,  
  fare INT,  
  FOREIGN KEY(user_id) REFERENCES users(id),  
  FOREIGN KEY(driver_id) REFERENCES drivers(id)  
);
```

-- Seed 4 drivers

```
INSERT INTO drivers (name,phone,location) VALUES  
('Arun','7777000001','Central Bus Stand'),  
('Kumar','7777000002','Chathiram Bus Stand'),  
('Aravind','7777000003','Srirangam'),  
('Ram','7777000004','TVS Tollgate');  
ALTER TABLE rides ADD COLUMN status VARCHAR(20) DEFAULT;
```

Python (Flask - mysqlconnector):

```
from flask import Flask, render_template, request, redirect, url_for, session, flash
import mysql.connector
from datetime import datetime

app = Flask(__name__)
app.secret_key = 'secret123'

db = mysql.connector.connect(
    host="localhost", user="root", password="root", database="cab_system"
)
cursor = db.cursor(dictionary=True)

FARE = {
    'Central Bus Stand': 100,
    'Chathiram Bus Stand': 150,
    'Srirangam': 200,
    'TVS Tollgate': 150
}

# Home / Help
@app.route('/')
def home():
    return render_template('home.html')

@app.route('/help')
def help():
    return render_template('help.html')

# Register
@app.route('/register', methods=['GET', 'POST'])
def register():
    if request.method == 'POST':
        n, e, p, ph = (request.form[k] for k in ('name', 'email', 'password', 'phone'))
        cursor.execute("SELECT 1 FROM users WHERE email=%s", (e,))
        if cursor.fetchone():
            flash("Email already registered", "danger")
        else:
            cursor.execute(
```

```

        "INSERT INTO users(name,email,password,phone)
VALUES(%s,%s,%s,%s)",
        (n,e,p,ph)
    ); db.commit()
    flash("Registered! Please log in.,"success")
    return redirect(url_for('user_login'))
    return render_template('register.html')

# User Login / Logout
@app.route('/user_login', methods=['GET','POST'])
def user_login():
    if request.method=='POST':
        e,p = request.form['email'],request.form['password']
        cursor.execute("SELECT * FROM users WHERE email=%s AND
password=%s",(e,p))
        u = cursor.fetchone()
        if u:
            session['user_id'], session['user_name'] = u['id'], u['name']
            return redirect(url_for('user_dashboard'))
            flash("Invalid credentials","danger")
            return render_template('user_login.html')

@app.route('/user_logout')
def user_logout():
    session.clear()
    return redirect(url_for('home'))

# User Dashboard / Request Ride
@app.route('/user_dashboard', methods=['GET','POST'])
def user_dashboard():
    if 'user_id' not in session:
        return redirect(url_for('user_login'))

    # get wallet
    cursor.execute("SELECT wallet FROM users WHERE
id=%s",(session['user_id'],))
    wallet = cursor.fetchone()['wallet']

    # live availability
    cursor.execute("SELECT location,name FROM drivers WHERE
is_available=1")

```

```

avail = {r['location']: r['name'] for r in cursor.fetchall()}

if request.method=='POST':
    pickup = request.form['pickup']
    time = request.form['time']
    fare = FARE[pickup]
    cursor.execute(
        "INSERT INTO rides(user_id,pickup,time,fare)
VALUES(%s,%s,%s,%s)",
        (session['user_id'],pickup,time,fare)
    ); db.commit()
    flash(f'Requested ride for ₹{fare}.',"info")
    return redirect(url_for('user_history'))

return render_template('user_dashboard.html',
                        fares=FARE, avail=avail, wallet=wallet)

# User History / Complete / Cancel
@app.route('/user_history')
def user_history():
    if 'user_id' not in session:
        return redirect(url_for('user_login'))
    # wallet
    cursor.execute("SELECT wallet FROM users WHERE
id=%s",(session['user_id'],))
    wallet = cursor.fetchone()['wallet']

    cursor.execute("""
    SELECT r.*, d.name driver_name, d.phone driver_phone
    FROM rides r
    LEFT JOIN drivers d ON d.id=r.driver_id
    WHERE r.user_id=%s
    ORDER BY r.id DESC
    """,(session['user_id'],))
    rides = cursor.fetchall()
    return render_template('user_history.html',
                            rides=rides, wallet=wallet)

@app.route('/user_complete/<int:ride_id>')
def user_complete(ride_id):
    if 'user_id' not in session:

```



```

        return redirect(url_for('user_login'))
# Only assigned→completed
cursor.execute("""
    UPDATE rides
    SET status='completed'
    WHERE id=%s AND user_id=%s AND status='assigned'
    """,(ride_id,session['user_id']))
# driver earnings +100
cursor.execute("SELECT driver_id FROM rides WHERE id=%s",(ride_id,))
d = cursor.fetchone()['driver_id']
if d:
    cursor.execute("UPDATE drivers SET earnings=earnings+100 WHERE
id=%s",(d,))
    db.commit()
    flash("Ride completed.", "success")
    return redirect(url_for('user_history'))

@app.route('/user_cancel/<int:ride_id>')
def user_cancel(ride_id):
    if 'user_id' not in session:
        return redirect(url_for('user_login'))
    cursor.execute("SELECT status,fare,driver_id FROM rides WHERE id=%s
AND user_id=%s",
                    (ride_id,session['user_id']))
    row = cursor.fetchone()
    if not row or row['status'] in ('completed','cancelled'):
        flash("Cannot cancel.", "warning")
    else:
        status,fare,drv = row['status'],row['fare'],row['driver_id']
        if status=='pending':
            refund = fare
        else:
            refund = fare//2
            # driver gets 50
            if drv:
                cursor.execute("UPDATE drivers SET earnings=earnings+50
WHERE id=%s",(drv,))
            cursor.execute("UPDATE users SET wallet=wallet+%s WHERE
id=%s",(refund,session['user_id']))
            cursor.execute("UPDATE rides SET status='cancelled' WHERE
id=%s",(ride_id,))

```

```

        db.commit()
        flash(f'Cancelled. ₹{refund} refunded.', "info")
        return redirect(url_for('user_history'))

# Driver Login / Logout
@app.route('/driver_login', methods=['GET','POST'])
def driver_login():
    if request.method=='POST':
        ph = request.form['phone']
        cursor.execute("SELECT * FROM drivers WHERE phone=%s",(ph,))
        d = cursor.fetchone()
        if d:
            session['driver_id'], session['driver_name'] = d['id'], d['name']
            return redirect(url_for('driver_dashboard'))
        flash("Driver not found","danger")
    return render_template('driver_login.html')

@app.route('/driver_logout')
def driver_logout():
    session.clear()
    return redirect(url_for('home'))

# Driver Dashboard / Assign / Complete
@app.route('/driver_dashboard', methods=['GET','POST'])
def driver_dashboard():
    if 'driver_id' not in session:
        return redirect(url_for('driver_login'))

    if request.method=='POST':
        if 'toggle' in request.form:
            cursor.execute("UPDATE drivers SET is_available=NOT is_available
WHERE id=%s",
                (session['driver_id'],)); db.commit()
            flash("Availability toggled.", "info")
        if 'assign' in request.form:
            cursor.execute("SELECT 1 FROM rides WHERE driver_id=%s AND
status='assigned'",
                (session['driver_id'],))
            if cursor.fetchone():
                flash("Finish current ride.", "warning")
            else:

```

```

        rid = request.form['assign']
        cursor.execute("UPDATE rides SET driver_id=%s,status='assigned'
WHERE id=%s",
                        (session['driver_id'],rid)); db.commit()
        flash("Ride assigned.", "success")
    if 'complete' in request.form:
        rid = request.form['complete']
        cursor.execute("UPDATE rides SET status='completed' WHERE id=%s
AND driver_id=%s",
                        (rid,session['driver_id']))
        cursor.execute("UPDATE drivers SET earnings=earnings+100 WHERE
id=%s",
                        (session['driver_id'],))
        db.commit()
        flash("Ride completed.", "success")

```

Pending

```

cursor.execute("""
SELECT r.id,u.name user_name,u.phone user_phone,r.pickup,r.time,r.fare
FROM rides r JOIN users u ON u.id=r.user_id
WHERE r.status='pending'
ORDER BY r.time
""")
pending = cursor.fetchall()

```

Current

```

cursor.execute("""
SELECT r.id,u.name user_name,r.pickup,r.time,r.fare
FROM rides r JOIN users u ON u.id=r.user_id
WHERE r.driver_id=%s AND r.status='assigned'
""",(session['driver_id'],))
current = cursor.fetchone()

```

Availability

```

cursor.execute("SELECT is_available,earnings FROM drivers WHERE
id=%s",
                (session['driver_id'],))
info = cursor.fetchone()
available, earnings = info['is_available'], info['earnings']

```

```

return render_template('driver_dashboard.html',

```

```

        pending=pending,
        current=current,
        available=available,
        earnings=earnings)

# Driver History & Earnings
@app.route('/driver_history')
def driver_history():
    if 'driver_id' not in session:
        return redirect(url_for('driver_login'))

    cursor.execute("""
        SELECT r.id, r.pickup, r.time, r.fare, r.status,
        CASE
            WHEN r.status='completed' THEN 100
            WHEN r.status='cancelled' THEN 50
            ELSE 0
        END AS earned
        FROM rides r
        WHERE r.driver_id=%s
        ORDER BY r.time DESC
    """,(session['driver_id'],))
    history = cursor.fetchall()

    total = sum(r['earned'] for r in history)
    return render_template('driver_history.html',
        history=history,
        total=total)

# Admin Login / Dashboard
@app.route('/admin_login', methods=['GET','POST'])
def admin_login():
    if request.method=='POST':
        u,p = request.form['username'],request.form['password']
        if u=='admin' and p=='admin@123':
            session['admin']=True
            return redirect(url_for('admin_dashboard'))
        flash("Invalid admin","danger")
    return render_template('admin_login.html')

```

```

@app.route('/admin_dashboard', methods=['GET','POST'])
def admin_dashboard():
    if not session.get('admin'):
        return redirect(url_for('admin_login'))

    if request.method=='POST' and 'update_user' in request.form:
        uid,n,e,ph = (request.form[k] for k in ('u_id','u_name','u_email','u_phone'))
        cursor.execute("UPDATE users SET name=%s,email=%s,phone=%s
WHERE id=%s",
                        (n,e,ph,uid)); db.commit()
        flash("User updated","success")
    if request.method=='POST' and 'add_driver' in request.form:
        dn,dp,dl = (request.form[k] for k in ('d_name','d_phone','d_location'))
        cursor.execute("INSERT INTO drivers(name,phone,location)
VALUES(%s,%s,%s)",
                        (dn,dp,dl)); db.commit()
        flash("Driver added","success")

    cursor.execute("SELECT * FROM users"); users = cursor.fetchall()
    cursor.execute("SELECT * FROM drivers"); drivers = cursor.fetchall()
    cursor.execute("""
    SELECT r.id, u.name AS user_name, d.name AS driver_name,
           r.pickup, r.time, r.fare, r.status
    FROM rides r
    LEFT JOIN users u ON u.id=r.user_id
    LEFT JOIN drivers d ON d.id=r.driver_id
    ORDER BY r.id DESC
    """); rides = cursor.fetchall()

    return render_template('admin_dashboard.html',
                           users=users,
                           drivers=drivers,
                           rides=rides)

@app.route('/logout')
def logout():
    session.clear()
    return redirect(url_for('home'))

if __name__ == '__main__':
    app.run(debug=True)

```

APPENDIX B - SCREENSHOTS

User Login:

The screenshot shows a web browser window with the title "KRCT Cab Booking". The address bar displays "127.0.0.1:5000/user_login". The page has a green header bar with the text "KRCT Cab Booking" on the left and "Home Login Help" on the right. The main content area is light gray and contains a white login form. The form has a title "User Login" and two input fields: "Email" with the value "vaithyanadhan.ad23@krct.ac.in" and "Password" with a masked value "...". A green "Login" button is at the bottom of the form. A small error message "Please fill out this field." is visible next to the password field.

User Dashboard:

The screenshot shows a web browser window with the title "KRCT Cab Booking". The address bar displays "127.0.0.1:5000/user_dashboard". The page has a green header bar with the text "KRCT Cab Booking" on the left and "Home Dashboard History Logout Help" on the right. The main content area is light gray and contains a white dashboard. At the top left of the dashboard, it says "Hi, vaithi (Wallet: ₹0)". In the center is a white box titled "Request a Ride" with a "Pickup" dropdown menu showing "Central Bus Stand", a "Date & Time" input field showing "30-05-2025 22:31", and a green "Request Ride" button. At the bottom left, under the heading "Available Drivers", there is a list item "Central Bus Stand — Arun".

User Ride:

The screenshot shows a web browser window with the title 'KRCT Cab Booking'. The address bar displays '127.0.0.1:5000/user_history'. The page has a green header with the text 'KRCT Cab Booking' on the left and navigation links 'Home', 'Dashboard', 'History', 'Logout', and 'Help' on the right. The main content area is titled 'Your Rides (Wallet: ₹0)'. Below this title is a table with the following data:

ID	Pickup	Time	Fare	Status	Driver	Action
1	Central Bus Stand	2025-05-24 23:33:00	₹100	completed	Arun	✓

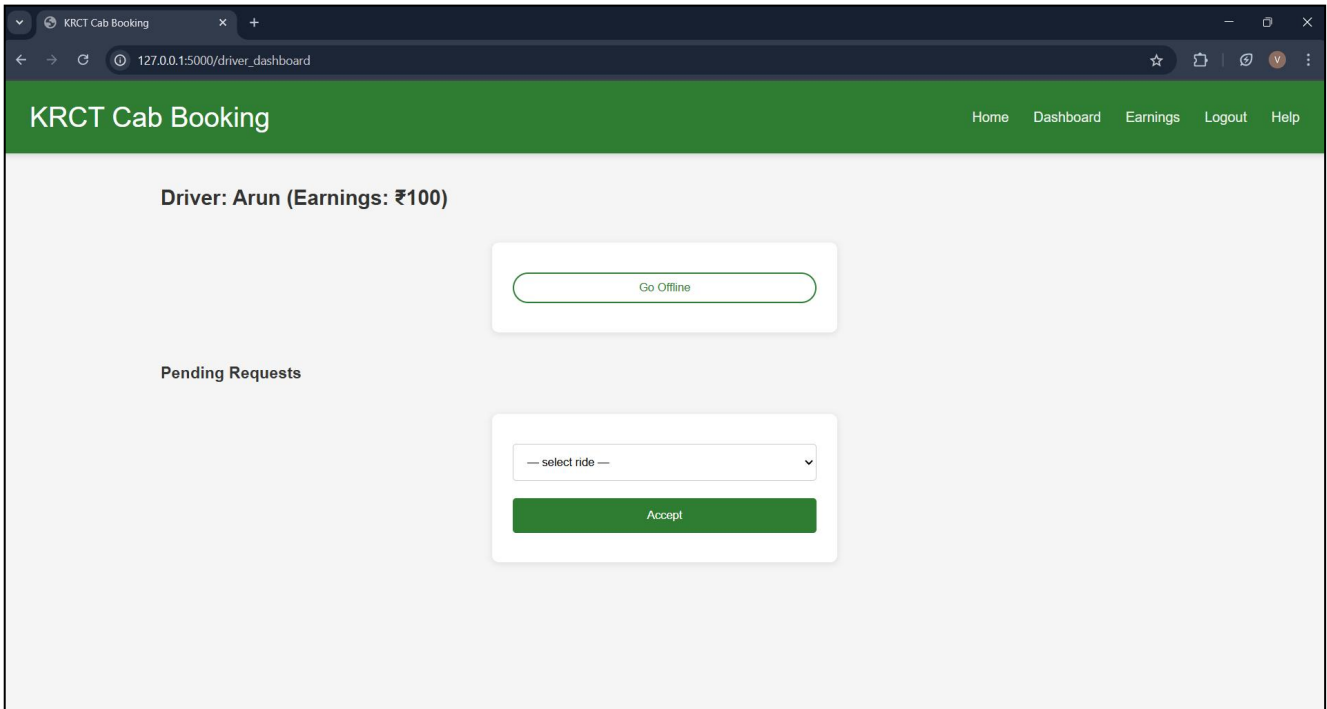
Driver Login:

The screenshot shows a web browser window with the title 'KRCT Cab Booking'. The address bar displays '127.0.0.1:5000/driver_login'. The page has a green header with the text 'KRCT Cab Booking' on the left and navigation links 'Home', 'Login', and 'Help' on the right. The main content area features a 'Driver Login' form with the following fields and buttons:

Driver Login

Phone

Driver Ride Checking:



The screenshot shows the 'KRCT Cab Booking' driver dashboard. At the top, a green header bar contains the app name and navigation links: Home, Dashboard, Earnings, Logout, and Help. Below the header, the main content area is light gray. It starts with the text 'Driver: Arun (Earnings: ₹100)'. To the right of this text is a white button with a green border labeled 'Go Offline'. Below this, the section 'Pending Requests' is shown. It contains a white box with a dropdown menu labeled '— select ride —' and a green 'Accept' button below it.

KRCT Cab Booking

Home Dashboard Earnings Logout Help

Driver: Arun (Earnings: ₹100)

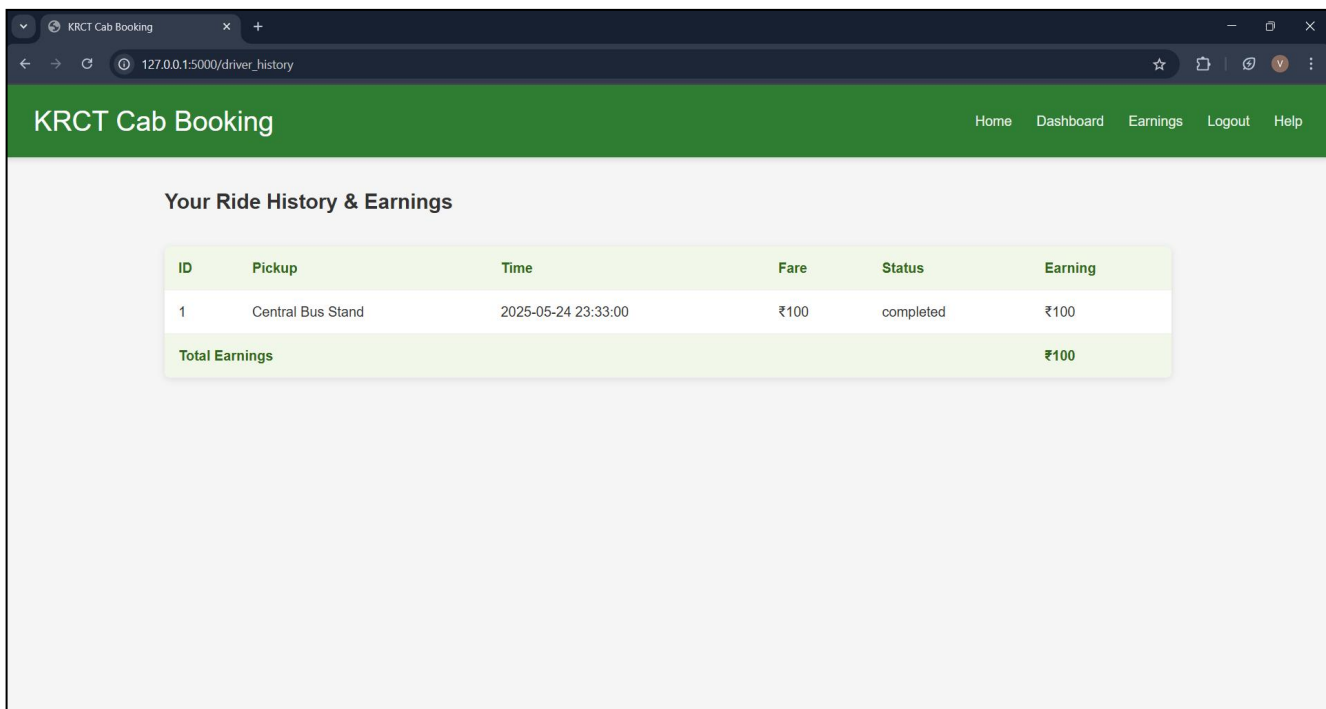
Go Offline

Pending Requests

— select ride —

Accept

Driver Fair:



The screenshot shows the 'KRCT Cab Booking' driver history page. It features a green header bar with the app name and navigation links: Home, Dashboard, Earnings, Logout, and Help. The main content area is light gray and titled 'Your Ride History & Earnings'. Below the title is a table with columns: ID, Pickup, Time, Fare, Status, and Earning. The table contains one row of ride data and a summary row for 'Total Earnings'.

KRCT Cab Booking

Home Dashboard Earnings Logout Help

Your Ride History & Earnings

ID	Pickup	Time	Fare	Status	Earning
1	Central Bus Stand	2025-05-24 23:33:00	₹100	completed	₹100
Total Earnings					₹100

Admin Login:

KRCT Cab Booking

Home Login Help

Admin Login

Username

admin

Password

Login

Admin Dashboard:

KRCT Cab Booking

Home Admin Logout Help

Admin Dashboard

Edit Users

ID	Name	Email	Phone	Save
1	vai	vai	978	Save

Add Driver

Name

Phone

Central Bus Stand

Add Driver

REFERENCES

1. Flask Documentation. (n.d.). Flask: The Python Micro Framework. Retrieved from <https://flask.palletsprojects.com/>.
2. GeeksforGeeks. (n.d.): Flask Tutorials, Python Projects, and MySQL Integration. Retrieved from <https://www.geeksforgeeks.org/>.
3. Grinberg, M. (2018). Flask Web Development: Developing Web Applications with Python (2nd ed.). O'Reilly Media.
4. MySQL Documentation Team. (2013): MySQL Reference Manual. Oracle Corporation.
5. MySQL Documentation. (n.d.): MySQL 8.0 Reference Manual. Retrieved from <https://dev.mysql.com/doc/>.
6. Real Python. (n.d.): Flask Tutorials and Web Development Projects. Retrieved from <https://realpython.com/>.
7. W3Schools. (n.d.): HTML, CSS, JavaScript, and SQL Tutorials. Retrieved from <https://www.w3schools.com/>.