Project Structure

ccc\_emergency\_map/

├── app.py

├── config.py

├── models.py

├── forms.py

├── decorators.py

├── customer\_routes.py

├── employee\_routes.py

├── admin\_routes.py

├── superadmin\_routes.py

├── templates/

│ ├── base.html

│ ├── index.html

│ ├── login.html

│ ├── register.html

│ ├── customer/

│ │ ├── index.html

│ │ ├── view\_trips.html

│ │ ├── check\_in.html

│ │ ├── create\_distress.html

│ │ ├── view\_site\_details.html

│ │ └── leave\_review.html

│ ├── employee/

│ │ ├── index.html

│ │ ├── manage\_trips.html

│ │ ├── assigned\_alerts.html

│ │ ├── all\_distress\_trips.html

│ │ ├── view\_site\_details.html

│ │ └── customer\_reviews.html

│ ├── admin/

│ │ ├── index.html

│ │ ├── create\_employee.html

│ │ ├── view\_trips.html

│ │ ├── view\_site\_details.html

│ │ └── override\_assignments.html

│ ├── superadmin/

│ │ ├── index.html

│ │ ├── create\_user.html

│ │ ├── manage\_users.html

│ │ ├── create\_site\_details.html

│ │ └── manage\_site\_details.html

├── static/

│ ├── styles.css

├── requirements.txt

├── README.md

└── ccc\_emergency\_map\_schema.sql

File: app.py

from flask import Flask, render\_template, redirect, url\_for, flash

from config import Config

from models import db, login\_manager

from flask\_migrate import Migrate

from customer\_routes import customer\_blueprint

from employee\_routes import employee\_blueprint

from admin\_routes import admin\_blueprint

from superadmin\_routes import superadmin\_blueprint

from forms import LoginForm, RegistrationForm

from flask\_login import login\_user, logout\_user, login\_required, current\_user

app = Flask(\_\_name\_\_)

app.config.from\_object(Config)

# Initialize extensions

db.init\_app(app)

login\_manager.init\_app(app)

migrate = Migrate(app, db)

# Register Blueprints

app.register\_blueprint(customer\_blueprint, url\_prefix='/customer')

app.register\_blueprint(employee\_blueprint, url\_prefix='/employee')

app.register\_blueprint(admin\_blueprint, url\_prefix='/admin')

app.register\_blueprint(superadmin\_blueprint, url\_prefix='/superadmin')

# User Loader

@login\_manager.user\_loader

def load\_user(user\_id):

from models import Account

return Account.query.get(int(user\_id))

# Routes for authentication

@app.route('/')

def index():

return render\_template('index.html')

@app.route('/login', methods=['GET', 'POST'])

def login():

from models import Account

form = LoginForm()

if form.validate\_on\_submit():

user = Account.query.filter\_by(Email=form.email.data).first()

if user and user.check\_password(form.password.data):

login\_user(user)

flash('Logged in successfully.')

return redirect(url\_for('dashboard'))

else:

flash('Invalid email or password.')

return render\_template('login.html', form=form)

@app.route('/register', methods=['GET', 'POST'])

def register():

from models import Account, Customer

form = RegistrationForm()

if form.validate\_on\_submit():

if Account.query.filter\_by(Email=form.email.data).first():

flash('Email already registered.')

return redirect(url\_for('register'))

if Account.query.filter\_by(Username=form.username.data).first():

flash('Username already taken.')

return redirect(url\_for('register'))

user = Account(

Username=form.username.data,

Email=form.email.data,

FirstName=form.first\_name.data,

LastName=form.last\_name.data,

AccountType='Customer',

PermissionTier=1

)

user.set\_password(form.password.data)

db.session.add(user)

db.session.commit()

# Create customer profile

customer = Customer(CustomerID=user.AccountID)

db.session.add(customer)

db.session.commit()

flash('Registration successful. Please log in.')

return redirect(url\_for('login'))

return render\_template('register.html', form=form)

@app.route('/logout')

@login\_required

def logout():

logout\_user()

flash('You have been logged out.')

return redirect(url\_for('index'))

@app.route('/dashboard')

@login\_required

def dashboard():

if current\_user.PermissionTier == 1:

return redirect(url\_for('customer.index'))

elif current\_user.PermissionTier == 2:

return redirect(url\_for('employee.index'))

elif current\_user.PermissionTier == 3:

return redirect(url\_for('admin.index'))

elif current\_user.PermissionTier == 4:

return redirect(url\_for('superadmin.index'))

else:

flash('Unauthorized access.')

return redirect(url\_for('logout'))

if \_\_name\_\_ == '\_\_main\_\_':

app.run(debug=True)

File: config.py

import os

class Config:

SECRET\_KEY = os.environ.get('SECRET\_KEY') or 'your\_default\_secret\_key'

MYSQL\_HOST = os.environ.get('MYSQL\_HOST') or 'localhost'

MYSQL\_USER = os.environ.get('MYSQL\_USER') or 'root'

MYSQL\_PASSWORD = os.environ.get('MYSQL\_PASSWORD') or 'your\_mysql\_password'

MYSQL\_DB = os.environ.get('MYSQL\_DB') or 'ccc\_emergency\_map'

# Construct SQLALCHEMY\_DATABASE\_URI based on the above credentials

SQLALCHEMY\_DATABASE\_URI = (

f'mysql+pymysql://{MYSQL\_USER}:{MYSQL\_PASSWORD}@{MYSQL\_HOST}/{MYSQL\_DB}'

)

SQLALCHEMY\_TRACK\_MODIFICATIONS = False

File: models.py

from flask\_sqlalchemy import SQLAlchemy

from flask\_login import UserMixin, LoginManager

from werkzeug.security import generate\_password\_hash, check\_password\_hash

from datetime import datetime

from sqlalchemy.dialects.mysql import INTEGER, VARCHAR, TEXT, ENUM, DECIMAL, TINYINT, TIMESTAMP, DATETIME

from sqlalchemy import ForeignKey, UniqueConstraint, CheckConstraint

db = SQLAlchemy()

login\_manager = LoginManager()

# Permissions Model

class Permissions(db.Model):

\_\_tablename\_\_ = 'Permissions'

PermissionID = db.Column(TINYINT(unsigned=True), primary\_key=True)

PermissionName = db.Column(VARCHAR(100), unique=True, nullable=False)

# Account Model

class Account(UserMixin, db.Model):

\_\_tablename\_\_ = 'Account'

AccountID = db.Column(INTEGER(unsigned=True), primary\_key=True, autoincrement=True)

Username = db.Column(VARCHAR(100), unique=True, nullable=False)

PasswordHash = db.Column(VARCHAR(255), nullable=False)

Email = db.Column(VARCHAR(100), unique=True, nullable=False)

PhoneNumber = db.Column(VARCHAR(15))

Address = db.Column(TEXT)

RegistrationDate = db.Column(TIMESTAMP, default=datetime.utcnow)

FirstName = db.Column(VARCHAR(100), nullable=False)

LastName = db.Column(VARCHAR(100), nullable=False)

AccountType = db.Column(ENUM('Customer', 'Employee', 'Admin', 'Super Admin'), nullable=False)

PermissionTier = db.Column(TINYINT(unsigned=True), ForeignKey('Permissions.PermissionID'), nullable=False, default=0)

# Relationships

customer = db.relationship('Customer', backref='account', uselist=False)

employee = db.relationship('Employee', backref='account', uselist=False)

def get\_id(self):

return self.AccountID

def set\_password(self, password):

self.PasswordHash = generate\_password\_hash(password)

def check\_password(self, password):

return check\_password\_hash(self.PasswordHash, password)

# Customer Model

class Customer(db.Model):

\_\_tablename\_\_ = 'Customer'

CustomerID = db.Column(INTEGER(unsigned=True), ForeignKey('Account.AccountID'), primary\_key=True)

EmergencyContactName = db.Column(VARCHAR(100))

EmergencyContactPhone = db.Column(VARCHAR(15))

# Relationships

trips = db.relationship('CustomerTrip', backref='customer', lazy=True)

feedbacks = db.relationship('CustomerFeedback', backref='customer', lazy=True)

# Employee Model

class Employee(db.Model):

\_\_tablename\_\_ = 'Employee'

EmployeeID = db.Column(INTEGER(unsigned=True), ForeignKey('Account.AccountID'), primary\_key=True)

Position = db.Column(VARCHAR(100), nullable=False)

HireDate = db.Column(TIMESTAMP, default=datetime.utcnow)

Status = db.Column(VARCHAR(50), default='Available', nullable=False)

RoleUpdateRequired = db.Column(TINYINT, default=0)

# Relationships

assigned\_alerts = db.relationship('EmergencyDistressAlerts', backref='assigned\_employee', lazy=True)

reported\_conditions = db.relationship('RouteConditionLog', backref='reported\_by\_employee', lazy=True)

# LocationsOfInterest Model

class LocationsOfInterest(db.Model):

\_\_tablename\_\_ = 'LocationsOfInterest'

LocationID = db.Column(INTEGER(unsigned=True), primary\_key=True, autoincrement=True)

Name = db.Column(VARCHAR(100), nullable=False)

Description = db.Column(TEXT)

Latitude = db.Column(DECIMAL(10, 7), nullable=False)

Longitude = db.Column(DECIMAL(10, 7), nullable=False)

Type = db.Column(VARCHAR(50))

Elevation = db.Column(DECIMAL(8, 2))

TerrainType = db.Column(VARCHAR(50))

AccessibilityNotes = db.Column(TEXT)

LastUpdated = db.Column(TIMESTAMP, default=datetime.utcnow, onupdate=datetime.utcnow)

# Relationships

start\_routes = db.relationship('Route', backref='start\_location', foreign\_keys='Route.StartLocationID', lazy=True)

end\_routes = db.relationship('Route', backref='end\_location', foreign\_keys='Route.EndLocationID', lazy=True)

resources = db.relationship('EmergencyResource', backref='current\_location', lazy=True)

# Route Model

class Route(db.Model):

\_\_tablename\_\_ = 'Route'

RouteID = db.Column(INTEGER(unsigned=True), primary\_key=True, autoincrement=True)

StartLocationID = db.Column(INTEGER(unsigned=True), ForeignKey('LocationsOfInterest.LocationID'), nullable=False)

EndLocationID = db.Column(INTEGER(unsigned=True), ForeignKey('LocationsOfInterest.LocationID'), nullable=False)

Distance = db.Column(DECIMAL(5, 2), nullable=False)

EstimatedTime = db.Column(DECIMAL(5, 2))

DifficultyLevel = db.Column(VARCHAR(50))

AgeRequirement = db.Column(INTEGER)

# Relationships

trips = db.relationship('Trip', backref='route', lazy=True)

condition\_logs = db.relationship('RouteConditionLog', backref='route', lazy=True)

points\_of\_interest = db.relationship('PointsOfInterest', backref='route', lazy=True)

# PointsOfInterest Model

class PointsOfInterest(db.Model):

\_\_tablename\_\_ = 'PointsOfInterest'

POIID = db.Column(INTEGER(unsigned=True), primary\_key=True, autoincrement=True)

Name = db.Column(VARCHAR(100), nullable=False)

Description = db.Column(TEXT)

Latitude = db.Column(DECIMAL(10, 7), nullable=False)

Longitude = db.Column(DECIMAL(10, 7), nullable=False)

RouteID = db.Column(INTEGER(unsigned=True), ForeignKey('Route.RouteID'), nullable=False)

Type = db.Column(VARCHAR(50))

# Relationships

geofences = db.relationship('Geofences', backref='poi', lazy=True)

# Geofences Model

class Geofences(db.Model):

\_\_tablename\_\_ = 'Geofences'

GeofenceID = db.Column(INTEGER(unsigned=True), primary\_key=True, autoincrement=True)

POIID = db.Column(INTEGER(unsigned=True), ForeignKey('PointsOfInterest.POIID'), nullable=False)

Radius = db.Column(DECIMAL(6, 2), nullable=False)

AlertType = db.Column(VARCHAR(50))

# TripType Model

class TripType(db.Model):

\_\_tablename\_\_ = 'TripType'

TripTypeID = db.Column(INTEGER(unsigned=True), primary\_key=True, autoincrement=True)

TypeName = db.Column(VARCHAR(100), unique=True, nullable=False)

Description = db.Column(TEXT)

# Relationships

trips = db.relationship('Trip', backref='trip\_type', lazy=True)

# Trip Model

class Trip(db.Model):

\_\_tablename\_\_ = 'Trip'

TripID = db.Column(INTEGER(unsigned=True), primary\_key=True, autoincrement=True)

RouteID = db.Column(INTEGER(unsigned=True), ForeignKey('Route.RouteID'), nullable=False)

TripTypeID = db.Column(INTEGER(unsigned=True), ForeignKey('TripType.TripTypeID'), nullable=False)

TripDate = db.Column(TIMESTAMP, default=datetime.utcnow)

TotalDistance = db.Column(DECIMAL(5, 2))

# Relationships

customer\_trips = db.relationship('CustomerTrip', backref='trip', lazy=True)

distress\_alerts = db.relationship('EmergencyDistressAlerts', backref='trip', lazy=True)

# CustomerTrip Model

class CustomerTrip(db.Model):

\_\_tablename\_\_ = 'CustomerTrip'

CustomerTripID = db.Column(INTEGER(unsigned=True), primary\_key=True, autoincrement=True)

CustomerID = db.Column(INTEGER(unsigned=True), ForeignKey('Customer.CustomerID'), nullable=False)

TripID = db.Column(INTEGER(unsigned=True), ForeignKey('Trip.TripID'), nullable=False)

\_\_table\_args\_\_ = (UniqueConstraint('CustomerID', 'TripID', name='\_customer\_trip\_uc'),)

# Relationships

feedbacks = db.relationship('CustomerFeedback', backref='customer\_trip', lazy=True)

alerts = db.relationship('EmergencyDistressAlerts', backref='customer\_trip', lazy=True)

# EmergencyDistressAlerts Model

class EmergencyDistressAlerts(db.Model):

\_\_tablename\_\_ = 'EmergencyDistressAlerts'

AlertID = db.Column(INTEGER(unsigned=True), primary\_key=True, autoincrement=True)

CustomerTripID = db.Column(INTEGER(unsigned=True), ForeignKey('CustomerTrip.CustomerTripID'), nullable=False)

Latitude = db.Column(DECIMAL(10, 7), nullable=False)

Longitude = db.Column(DECIMAL(10, 7), nullable=False)

Timestamp = db.Column(TIMESTAMP, default=datetime.utcnow)

Status = db.Column(VARCHAR(50), nullable=False)

SeverityLevel = db.Column(ENUM('Low', 'Medium', 'High', 'Critical'), default='Medium', nullable=False)

ResponseTime = db.Column(DATETIME)

ResolutionTime = db.Column(DATETIME)

ResolutionNotes = db.Column(TEXT)

AssignedEmployeeID = db.Column(INTEGER(unsigned=True), ForeignKey('Employee.EmployeeID'))

# CustomerFeedback Model

class CustomerFeedback(db.Model):

\_\_tablename\_\_ = 'CustomerFeedback'

FeedbackID = db.Column(INTEGER(unsigned=True), primary\_key=True, autoincrement=True)

CustomerTripID = db.Column(INTEGER(unsigned=True), ForeignKey('CustomerTrip.CustomerTripID'), nullable=False)

Feedback = db.Column(TEXT)

Rating = db.Column(INTEGER)

Timestamp = db.Column(TIMESTAMP, default=datetime.utcnow)

\_\_table\_args\_\_ = (

CheckConstraint('Rating BETWEEN 1 AND 5', name='check\_rating'),

)

# EmergencyDistressAssignmentQueue Model

class EmergencyDistressAssignmentQueue(db.Model):

\_\_tablename\_\_ = 'EmergencyDistressAssignmentQueue'

QueueID = db.Column(INTEGER(unsigned=True), primary\_key=True, autoincrement=True)

AlertID = db.Column(INTEGER(unsigned=True), ForeignKey('EmergencyDistressAlerts.AlertID'), nullable=False)

CreatedAt = db.Column(TIMESTAMP, default=datetime.utcnow)

Processed = db.Column(TINYINT, default=0)

# EmergencyResource Model

class EmergencyResource(db.Model):

\_\_tablename\_\_ = 'EmergencyResource'

ResourceID = db.Column(INTEGER(unsigned=True), primary\_key=True, autoincrement=True)

ResourceName = db.Column(VARCHAR(100), nullable=False)

ResourceType = db.Column(ENUM('Vehicle', 'Equipment', 'Personnel'), nullable=False)

CurrentLocationID = db.Column(INTEGER(unsigned=True), ForeignKey('LocationsOfInterest.LocationID'))

Status = db.Column(ENUM('Available', 'In Use', 'Out of Service'), default='Available')

LastUpdated = db.Column(TIMESTAMP, default=datetime.utcnow, onupdate=datetime.utcnow)

# EmergencyResponseTeam Model

class EmergencyResponseTeam(db.Model):

\_\_tablename\_\_ = 'EmergencyResponseTeam'

TeamID = db.Column(INTEGER(unsigned=True), primary\_key=True, autoincrement=True)

TeamName = db.Column(VARCHAR(100), nullable=False)

LeadEmployeeID = db.Column(INTEGER(unsigned=True), ForeignKey('Employee.EmployeeID'))

# Relationships

members = db.relationship('EmergencyResponseTeamMember', backref='team', lazy=True)

# EmergencyResponseTeamMember Model

class EmergencyResponseTeamMember(db.Model):

\_\_tablename\_\_ = 'EmergencyResponseTeamMember'

TeamID = db.Column(INTEGER(unsigned=True), ForeignKey('EmergencyResponseTeam.TeamID'), primary\_key=True)

EmployeeID = db.Column(INTEGER(unsigned=True), ForeignKey('Employee.EmployeeID'), primary\_key=True)

Role = db.Column(VARCHAR(50))

# RouteConditionLog Model

class RouteConditionLog(db.Model):

\_\_tablename\_\_ = 'RouteConditionLog'

LogID = db.Column(INTEGER(unsigned=True), primary\_key=True, autoincrement=True)

RouteID = db.Column(INTEGER(unsigned=True), ForeignKey('Route.RouteID'), nullable=False)

ConditionType = db.Column(ENUM('Normal', 'Caution', 'Danger', 'Closed'), nullable=False)

Description = db.Column(TEXT)

ReportedBy = db.Column(INTEGER(unsigned=True), ForeignKey('Employee.EmployeeID'))

ReportedAt = db.Column(TIMESTAMP, default=datetime.utcnow)

ResolvedAt = db.Column(TIMESTAMP)

# EmergencyIncidentAnalysis Model

class EmergencyIncidentAnalysis(db.Model):

\_\_tablename\_\_ = 'EmergencyIncidentAnalysis'

AnalysisID = db.Column(INTEGER(unsigned=True), primary\_key=True, autoincrement=True)

AlertID = db.Column(INTEGER(unsigned=True), ForeignKey('EmergencyDistressAlerts.AlertID'), nullable=False)

ResponseTime = db.Column(INTEGER) # in minutes

ResolutionTime = db.Column(INTEGER) # in minutes

SuccessFactors = db.Column(TEXT)

ImprovementAreas = db.Column(TEXT)

AnalyzedBy = db.Column(INTEGER(unsigned=True), ForeignKey('Employee.EmployeeID'))

AnalysisDate = db.Column(TIMESTAMP, default=datetime.utcnow)

# ActivityLog Model

class ActivityLog(db.Model):

\_\_tablename\_\_ = 'ActivityLog'

LogID = db.Column(INTEGER(unsigned=True), primary\_key=True, autoincrement=True)

ActivityType = db.Column(VARCHAR(100))

TableName = db.Column(VARCHAR(100))

RecordID = db.Column(VARCHAR(100))

OperationType = db.Column(VARCHAR(10))

Description = db.Column(TEXT)

Username = db.Column(VARCHAR(100))

Timestamp = db.Column(TIMESTAMP, default=datetime.utcnow)

File: forms.py

from flask\_wtf import FlaskForm

from wtforms import StringField, PasswordField, SubmitField, EmailField, IntegerField, SelectField, DecimalField, TextAreaField

from wtforms.validators import DataRequired, Email, EqualTo, NumberRange

class LoginForm(FlaskForm):

email = EmailField('Email', validators=[DataRequired(), Email()])

password = PasswordField('Password', validators=[DataRequired()])

submit = SubmitField('Login')

class RegistrationForm(FlaskForm):

first\_name = StringField('First Name', validators=[DataRequired()])

last\_name = StringField('Last Name', validators=[DataRequired()])

username = StringField('Username', validators=[DataRequired()])

email = EmailField('Email', validators=[DataRequired(), Email()])

password = PasswordField('Password', validators=[DataRequired()])

password2 = PasswordField(

'Repeat Password', validators=[DataRequired(), EqualTo('password')]

)

submit = SubmitField('Register')

class CreateAdminForm(FlaskForm):

first\_name = StringField('First Name', validators=[DataRequired()])

last\_name = StringField('Last Name', validators=[DataRequired()])

username = StringField('Username', validators=[DataRequired()])

email = EmailField('Email', validators=[DataRequired(), Email()])

account\_type = SelectField('Account Type', choices=[('Admin', 'Admin'), ('Super Admin', 'Super Admin')], validators=[DataRequired()])

permission\_tier = SelectField('Permission Tier', choices=[('3', 'Admin'), ('4', 'Super Admin')], validators=[DataRequired()])

password = PasswordField('Password', validators=[DataRequired()])

password2 = PasswordField(

'Repeat Password', validators=[DataRequired(), EqualTo('password')]

)

submit = SubmitField('Create Account')

class TripForm(FlaskForm):

route\_id = IntegerField('Route ID', validators=[DataRequired()])

trip\_type\_id = IntegerField('Trip Type ID', validators=[DataRequired()])

trip\_date = StringField('Trip Date', validators=[DataRequired()])

total\_distance = DecimalField('Total Distance', validators=[DataRequired()])

submit = SubmitField('Create Trip')

class DistressAlertForm(FlaskForm):

latitude = DecimalField('Latitude', validators=[DataRequired()])

longitude = DecimalField('Longitude', validators=[DataRequired()])

severity\_level = SelectField('Severity Level', choices=[('Low', 'Low'), ('Medium', 'Medium'), ('High', 'High'), ('Critical', 'Critical')], validators=[DataRequired()])

submit = SubmitField('Submit Alert')

class FeedbackForm(FlaskForm):

customer\_trip\_id = IntegerField('Customer Trip ID', validators=[DataRequired()])

rating = IntegerField('Rating', validators=[DataRequired(), NumberRange(min=1, max=5)])

feedback = TextAreaField('Feedback')

submit = SubmitField('Submit Review')

File: decorators.py

from functools import wraps

from flask import abort

from flask\_login import current\_user

def permission\_required(permission):

def decorator(f):

@wraps(f)

def decorated\_function(\*args, \*\*kwargs):

if current\_user.PermissionTier != permission:

abort(403)

return f(\*args, \*\*kwargs)

return decorated\_function

return decorator

File: customer\_routes.py

from flask import Blueprint, render\_template, redirect, url\_for, flash, request

from flask\_login import login\_required, current\_user

from decorators import permission\_required

from models import db, Trip, CustomerTrip, EmergencyDistressAlerts, CustomerFeedback, LocationsOfInterest

from forms import DistressAlertForm, FeedbackForm

customer\_blueprint = Blueprint('customer', \_\_name\_\_, template\_folder='templates/customer')

@customer\_blueprint.route('/')

@login\_required

@permission\_required(1)

def index():

return render\_template('customer/index.html')

@customer\_blueprint.route('/view\_trips')

@login\_required

@permission\_required(1)

def view\_trips():

trips = Trip.query.all()

return render\_template('customer/view\_trips.html', trips=trips)

@customer\_blueprint.route('/check\_in/<int:trip\_id>', methods=['GET', 'POST'])

@login\_required

@permission\_required(1)

def check\_in(trip\_id):

if request.method == 'POST':

# Check-in logic

customer\_trip = CustomerTrip(CustomerID=current\_user.AccountID, TripID=trip\_id)

db.session.add(customer\_trip)

db.session.commit()

flash('Checked in successfully.')

return redirect(url\_for('customer.view\_trips'))

trip = Trip.query.get\_or\_404(trip\_id)

return render\_template('customer/check\_in.html', trip=trip)

@customer\_blueprint.route('/create\_distress', methods=['GET', 'POST'])

@login\_required

@permission\_required(1)

def create\_distress():

form = DistressAlertForm()

if form.validate\_on\_submit():

# Assuming customer is on a trip

customer\_trip = CustomerTrip.query.filter\_by(CustomerID=current\_user.AccountID).order\_by(CustomerTrip.CustomerTripID.desc()).first()

if not customer\_trip:

flash('You are not checked into any trip.')

return redirect(url\_for('customer.index'))

alert = EmergencyDistressAlerts(

CustomerTripID=customer\_trip.CustomerTripID,

Latitude=form.latitude.data,

Longitude=form.longitude.data,

Status='Active',

SeverityLevel=form.severity\_level.data

)

db.session.add(alert)

db.session.commit()

flash('Distress alert created.')

return redirect(url\_for('customer.index'))

return render\_template('customer/create\_distress.html', form=form)

@customer\_blueprint.route('/view\_site\_details')

@login\_required

@permission\_required(1)

def view\_site\_details():

locations = LocationsOfInterest.query.all()

return render\_template('customer/view\_site\_details.html', locations=locations)

@customer\_blueprint.route('/leave\_review', methods=['GET', 'POST'])

@login\_required

@permission\_required(1)

def leave\_review():

form = FeedbackForm()

if form.validate\_on\_submit():

feedback = CustomerFeedback(

CustomerTripID=form.customer\_trip\_id.data,

Feedback=form.feedback.data,

Rating=form.rating.data

)

db.session.add(feedback)

db.session.commit()

flash('Review submitted.')

return redirect(url\_for('customer.index'))

# Fetch customer trips

customer\_trips = CustomerTrip.query.filter\_by(CustomerID=current\_user.AccountID).all()

return render\_template('customer/leave\_review.html', form=form, customer\_trips=customer\_trips)

File: employee\_routes.py

from flask import Blueprint, render\_template, redirect, url\_for, flash, request

from flask\_login import login\_required, current\_user

from decorators import permission\_required

from models import db, Trip, EmergencyDistressAlerts, CustomerFeedback, Route, LocationsOfInterest

from forms import TripForm

employee\_blueprint = Blueprint('employee', \_\_name\_\_, template\_folder='templates/employee')

@employee\_blueprint.route('/')

@login\_required

@permission\_required(2)

def index():

return render\_template('employee/index.html')

@employee\_blueprint.route('/manage\_trips', methods=['GET', 'POST'])

@login\_required

@permission\_required(2)

def manage\_trips():

form = TripForm()

if form.validate\_on\_submit():

# Create a new trip

trip = Trip(

RouteID=form.route\_id.data,

TripTypeID=form.trip\_type\_id.data,

TripDate=form.trip\_date.data,

TotalDistance=form.total\_distance.data

)

db.session.add(trip)

db.session.commit()

flash('Trip created.')

trips = Trip.query.all()

return render\_template('employee/manage\_trips.html', trips=trips, form=form)

@employee\_blueprint.route('/assigned\_alerts')

@login\_required

@permission\_required(2)

def assigned\_alerts():

alerts = EmergencyDistressAlerts.query.filter\_by(AssignedEmployeeID=current\_user.AccountID).all()

return render\_template('employee/assigned\_alerts.html', alerts=alerts)

@employee\_blueprint.route('/all\_distress\_trips')

@login\_required

@permission\_required(2)

def all\_distress\_trips():

alerts = EmergencyDistressAlerts.query.filter\_by(Status='Active').all()

return render\_template('employee/all\_distress\_trips.html', alerts=alerts)

@employee\_blueprint.route('/resolve\_alert/<int:alert\_id>', methods=['POST'])

@login\_required

@permission\_required(2)

def resolve\_alert(alert\_id):

alert = EmergencyDistressAlerts.query.get\_or\_404(alert\_id)

if alert.AssignedEmployeeID != current\_user.AccountID:

flash('Unauthorized action.')

return redirect(url\_for('employee.assigned\_alerts'))

alert.Status = 'Resolved'

db.session.commit()

flash('Alert resolved.')

return redirect(url\_for('employee.assigned\_alerts'))

@employee\_blueprint.route('/assign\_alert/<int:alert\_id>', methods=['POST'])

@login\_required

@permission\_required(2)

def assign\_alert(alert\_id):

alert = EmergencyDistressAlerts.query.get\_or\_404(alert\_id)

alert.AssignedEmployeeID = current\_user.AccountID

db.session.commit()

flash('Alert assigned to you.')

return redirect(url\_for('employee.all\_distress\_trips'))

@employee\_blueprint.route('/view\_site\_details')

@login\_required

@permission\_required(2)

def view\_site\_details():

locations = LocationsOfInterest.query.all()

return render\_template('employee/view\_site\_details.html', locations=locations)

@employee\_blueprint.route('/customer\_reviews')

@login\_required

@permission\_required(2)

def customer\_reviews():

reviews = CustomerFeedback.query.all()

return render\_template('employee/customer\_reviews.html', reviews=reviews)

File: admin\_routes.py

from flask import Blueprint, render\_template, redirect, url\_for, flash, request

from flask\_login import login\_required, current\_user

from decorators import permission\_required

from models import db, Account, Employee, Trip, LocationsOfInterest, Permissions

from forms import TripForm

admin\_blueprint = Blueprint('admin', \_\_name\_\_, template\_folder='templates/admin')

@admin\_blueprint.route('/')

@login\_required

@permission\_required(3)

def index():

return render\_template('admin/index.html')

@admin\_blueprint.route('/create\_employee', methods=['GET', 'POST'])

@login\_required

@permission\_required(3)

def create\_employee():

if request.method == 'POST':

data = request.form

if Account.query.filter\_by(Email=data['email']).first():

flash('Email already registered.')

return redirect(url\_for('admin.create\_employee'))

if Account.query.filter\_by(Username=data['username']).first():

flash('Username already taken.')

return redirect(url\_for('admin.create\_employee'))

user = Account(

Username=data['username'],

Email=data['email'],

FirstName=data['first\_name'],

LastName=data['last\_name'],

AccountType='Employee',

PermissionTier=2

)

user.set\_password(data['password'])

db.session.add(user)

db.session.commit()

# Create employee profile

employee = Employee(EmployeeID=user.AccountID, Position=data['position'])

db.session.add(employee)

db.session.commit()

flash('Employee account created.')

return redirect(url\_for('admin.index'))

return render\_template('admin/create\_employee.html')

@admin\_blueprint.route('/view\_trips')

@login\_required

@permission\_required(3)

def view\_trips():

trips = Trip.query.all()

return render\_template('admin/view\_trips.html', trips=trips)

@admin\_blueprint.route('/view\_site\_details', methods=['GET', 'POST'])

@login\_required

@permission\_required(3)

def view\_site\_details():

if request.method == 'POST':

data = request.form

location = LocationsOfInterest.query.get\_or\_404(data['location\_id'])

location.Name = data['name']

location.Description = data['description']

db.session.commit()

flash('Site details updated.')

locations = LocationsOfInterest.query.all()

return render\_template('admin/view\_site\_details.html', locations=locations)

@admin\_blueprint.route('/override\_assignments')

@login\_required

@permission\_required(3)

def override\_assignments():

# Logic to override emergency team assignments

flash('Override assignments feature not implemented yet.')

return redirect(url\_for('admin.index'))

File: superadmin\_routes.py

from flask import Blueprint, render\_template, redirect, url\_for, flash, request

from flask\_login import login\_required, current\_user

from decorators import permission\_required

from models import db, Account, LocationsOfInterest, Permissions

from forms import CreateAdminForm

superadmin\_blueprint = Blueprint('superadmin', \_\_name\_\_, template\_folder='templates/superadmin')

@superadmin\_blueprint.route('/')

@login\_required

@permission\_required(4)

def index():

return render\_template('superadmin/index.html')

@superadmin\_blueprint.route('/create\_user', methods=['GET', 'POST'])

@login\_required

@permission\_required(4)

def create\_user():

form = CreateAdminForm()

if form.validate\_on\_submit():

if Account.query.filter\_by(Email=form.email.data).first():

flash('Email already registered.')

return redirect(url\_for('superadmin.create\_user'))

if Account.query.filter\_by(Username=form.username.data).first():

flash('Username already taken.')

return redirect(url\_for('superadmin.create\_user'))

permission\_tier = int(form.permission\_tier.data)

if permission\_tier not in [3, 4]:

flash('Invalid permission tier.')

return redirect(url\_for('superadmin.create\_user'))

user = Account(

Username=form.username.data,

Email=form.email.data,

FirstName=form.first\_name.data,

LastName=form.last\_name.data,

AccountType=form.account\_type.data,

PermissionTier=permission\_tier

)

user.set\_password(form.password.data)

db.session.add(user)

db.session.commit()

flash(f'{form.account\_type.data} account created.')

return redirect(url\_for('superadmin.manage\_users'))

return render\_template('superadmin/create\_user.html', form=form)

@superadmin\_blueprint.route('/manage\_users', methods=['GET', 'POST'])

@login\_required

@permission\_required(4)

def manage\_users():

if request.method == 'POST':

data = request.form

if data.get('action') == 'delete':

user = Account.query.get\_or\_404(data['user\_id'])

if user.PermissionTier >= current\_user.PermissionTier:

flash('Cannot delete a user with equal or higher permission.')

return redirect(url\_for('superadmin.manage\_users'))

db.session.delete(user)

db.session.commit()

flash('User account deleted.')

users = Account.query.filter(Account.PermissionTier >= 3).all()

return render\_template('superadmin/manage\_users.html', users=users)

@superadmin\_blueprint.route('/create\_site\_details', methods=['GET', 'POST'])

@login\_required

@permission\_required(4)

def create\_site\_details():

if request.method == 'POST':

data = request.form

location = LocationsOfInterest(

Name=data['name'],

Description=data['description'],

Latitude=data['latitude'],

Longitude=data['longitude']

)

db.session.add(location)

db.session.commit()

flash('Site details created.')

return redirect(url\_for('superadmin.manage\_site\_details'))

return render\_template('superadmin/create\_site\_details.html')

@superadmin\_blueprint.route('/manage\_site\_details', methods=['GET', 'POST'])

@login\_required

@permission\_required(4)

def manage\_site\_details():

if request.method == 'POST':

data = request.form

location = LocationsOfInterest.query.get\_or\_404(data['location\_id'])

location.Name = data['name']

location.Description = data['description']

db.session.commit()

flash('Site details updated.')

locations = LocationsOfInterest.query.all()

return render\_template('superadmin/manage\_site\_details.html', locations=locations)

File: templates/base.html

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>CCC Emergency Map System</title>

<link rel="stylesheet" href="{{ url\_for('static', filename='styles.css') }}">

</head>

<body>

<header>

<h1>CCC Emergency Map System</h1>

<nav>

<a href="{{ url\_for('index') }}">Home</a>

{% if current\_user.is\_authenticated %}

<span>Welcome, {{ current\_user.Username }}!</span>

<a href="{{ url\_for('dashboard') }}">Dashboard</a>

<a href="{{ url\_for('logout') }}">Logout</a>

{% else %}

<a href="{{ url\_for('login') }}">Login</a>

<a href="{{ url\_for('register') }}">Register</a>

{% endif %}

</nav>

</header>

<main>

{% with messages = get\_flashed\_messages() %}

{% if messages %}

<ul class="flashes">

{% for message in messages %}

<li>{{ message }}</li>

{% endfor %}

</ul>

{% endif %}

{% endwith %}

{% block content %}{% endblock %}

</main>

</body>

</html>

File: templates/index.html

{% extends 'base.html' %}

{% block content %}

<h2>Welcome to the CCC Emergency Map System</h2>

<p>Please log in or register to continue.</p>

{% endblock %}

File: templates/login.html

{% extends 'base.html' %}

{% block content %}

<h2>Login</h2>

<form method="post">

{{ form.hidden\_tag() }}

<label>Email:</label><br>

{{ form.email(size=32) }}<br><br>

<label>Password:</label><br>

{{ form.password(size=32) }}<br><br>

{{ form.submit() }}

</form>

{% endblock %}

File: templates/register.html

{% extends 'base.html' %}

{% block content %}

<h2>Register</h2>

<form method="post">

{{ form.hidden\_tag() }}

<label>First Name:</label><br>

{{ form.first\_name(size=32) }}<br><br>

<label>Last Name:</label><br>

{{ form.last\_name(size=32) }}<br><br>

<label>Username:</label><br>

{{ form.username(size=32) }}<br><br>

<label>Email:</label><br>

{{ form.email(size=32) }}<br><br>

<label>Password:</label><br>

{{ form.password(size=32) }}<br><br>

<label>Repeat Password:</label><br>

{{ form.password2(size=32) }}<br><br>

{{ form.submit() }}

</form>

{% endblock %}

File: templates/customer/index.html

{% extends 'base.html' %}

{% block content %}

<h2>Customer Dashboard</h2>

<ul>

<li><a href="{{ url\_for('customer.view\_trips') }}">View Trips</a></li>

<li><a href="{{ url\_for('customer.create\_distress') }}">Create Distress Alert</a></li>

<li><a href="{{ url\_for('customer.view\_site\_details') }}">View Site Details</a></li>

<li><a href="{{ url\_for('customer.leave\_review') }}">Leave a Review</a></li>

</ul>

{% endblock %}

File: templates/customer/view\_trips.html

{% extends 'base.html' %}

{% block content %}

<h2>Available Trips</h2>

<table>

<tr>

<th>Trip ID</th>

<th>Route</th>

<th>Date</th>

<th>Actions</th>

</tr>

{% for trip in trips %}

<tr>

<td>{{ trip.TripID }}</td>

<td>{{ trip.route.start\_location.Name }} to {{ trip.route.end\_location.Name }}</td>

<td>{{ trip.TripDate.strftime('%Y-%m-%d %H:%M') }}</td>

<td><a href="{{ url\_for('customer.check\_in', trip\_id=trip.TripID) }}">Check In</a></td>

</tr>

{% endfor %}

</table>

{% endblock %}

File: templates/customer/check\_in.html

{% extends 'base.html' %}

{% block content %}

<h2>Check In to Trip</h2>

<p>Are you sure you want to check in to <strong>{{ trip.route.start\_location.Name }} to {{ trip.route.end\_location.Name }}</strong> on {{ trip.TripDate.strftime('%Y-%m-%d %H:%M') }}?</p>

<form method="post">

<button type="submit">Yes, Check In</button>

<a href="{{ url\_for('customer.view\_trips') }}">Cancel</a>

</form>

{% endblock %}

File: templates/customer/create\_distress.html

{% extends 'base.html' %}

{% block content %}

<h2>Create Distress Alert</h2>

<form method="post">

{{ form.hidden\_tag() }}

<label>Latitude:</label><br>

{{ form.latitude() }}<br><br>

<label>Longitude:</label><br>

{{ form.longitude() }}<br><br>

<label>Severity Level:</label><br>

{{ form.severity\_level() }}<br><br>

{{ form.submit() }}

</form>

{% endblock %}

File: templates/customer/view\_site\_details.html

{% extends 'base.html' %}

{% block content %}

<h2>Site Details</h2>

{% for location in locations %}

<h3>{{ location.Name }}</h3>

<p>{{ location.Description }}</p>

<p><strong>Coordinates:</strong> {{ location.Latitude }}, {{ location.Longitude }}</p>

<hr>

{% endfor %}

{% endblock %}

File: templates/customer/leave\_review.html

{% extends 'base.html' %}

{% block content %}

<h2>Leave a Review</h2>

<form method="post">

{{ form.hidden\_tag() }}

<label>Customer Trip ID:</label><br>

<select name="customer\_trip\_id">

{% for trip in customer\_trips %}

<option value="{{ trip.CustomerTripID }}">{{ trip.Trip.TripID }} - {{ trip.Trip.route.start\_location.Name }} to {{ trip.Trip.route.end\_location.Name }}</option>

{% endfor %}

</select><br><br>

<label>Rating (1-5):</label><br>

{{ form.rating() }}<br><br>

<label>Feedback:</label><br>

{{ form.feedback(rows=4, cols=50) }}<br><br>

{{ form.submit() }}

</form>

{% endblock %}

File: templates/employee/index.html

{% extends 'base.html' %}

{% block content %}

<h2>Employee Dashboard</h2>

<ul>

<li><a href="{{ url\_for('employee.manage\_trips') }}">Manage Trips</a></li>

<li><a href="{{ url\_for('employee.assigned\_alerts') }}">Assigned Alerts</a></li>

<li><a href="{{ url\_for('employee.all\_distress\_trips') }}">All Distress Trips</a></li>

<li><a href="{{ url\_for('employee.view\_site\_details') }}">View Site Details</a></li>

<li><a href="{{ url\_for('employee.customer\_reviews') }}">Customer Reviews</a></li>

</ul>

{% endblock %}

File: templates/employee/manage\_trips.html

{% extends 'base.html' %}

{% block content %}

<h2>Manage Trips</h2>

<form method="post">

{{ form.hidden\_tag() }}

<label>Route ID:</label><br>

{{ form.route\_id() }}<br><br>

<label>Trip Type ID:</label><br>

{{ form.trip\_type\_id() }}<br><br>

<label>Trip Date (YYYY-MM-DD HH:MM):</label><br>

{{ form.trip\_date() }}<br><br>

<label>Total Distance:</label><br>

{{ form.total\_distance() }}<br><br>

{{ form.submit() }}

</form>

<h3>Existing Trips</h3>

<table>

<tr>

<th>Trip ID</th>

<th>Route</th>

<th>Date</th>

<th>Total Distance</th>

</tr>

{% for trip in trips %}

<tr>

<td>{{ trip.TripID }}</td>

<td>{{ trip.route.start\_location.Name }} to {{ trip.route.end\_location.Name }}</td>

<td>{{ trip.TripDate.strftime('%Y-%m-%d %H:%M') }}</td>

<td>{{ trip.TotalDistance }}</td>

</tr>

{% endfor %}

</table>

{% endblock %}

File: templates/employee/assigned\_alerts.html

{% extends 'base.html' %}

{% block content %}

<h2>Assigned Alerts</h2>

<table>

<tr>

<th>Alert ID</th>

<th>Customer Trip ID</th>

<th>Location</th>

<th>Severity</th>

<th>Status</th>

<th>Actions</th>

</tr>

{% for alert in alerts %}

<tr>

<td>{{ alert.AlertID }}</td>

<td>{{ alert.CustomerTripID }}</td>

<td>{{ alert.Latitude }}, {{ alert.Longitude }}</td>

<td>{{ alert.SeverityLevel }}</td>

<td>{{ alert.Status }}</td>

<td>

{% if alert.Status != 'Resolved' %}

<form method="post" action="{{ url\_for('employee.resolve\_alert', alert\_id=alert.AlertID) }}">

<button type="submit">Resolve</button>

</form>

{% else %}

<span>Resolved</span>

{% endif %}

</td>

</tr>

{% endfor %}

</table>

{% endblock %}

File: templates/employee/all\_distress\_trips.html

{% extends 'base.html' %}

{% block content %}

<h2>All Active Distress Alerts</h2>

<table>

<tr>

<th>Alert ID</th>

<th>Customer Trip ID</th>

<th>Location</th>

<th>Severity</th>

<th>Status</th>

<th>Actions</th>

</tr>

{% for alert in alerts %}

<tr>

<td>{{ alert.AlertID }}</td>

<td>{{ alert.CustomerTripID }}</td>

<td>{{ alert.Latitude }}, {{ alert.Longitude }}</td>

<td>{{ alert.SeverityLevel }}</td>

<td>{{ alert.Status }}</td>

<td>

{% if alert.AssignedEmployeeID is none %}

<form method="post" action="{{ url\_for('employee.assign\_alert', alert\_id=alert.AlertID) }}">

<button type="submit">Assign to Me</button>

</form>

{% else %}

<span>Assigned to {{ alert.assigned\_employee.Username }}</span>

{% endif %}

</td>

</tr>

{% endfor %}

</table>

{% endblock %}

File: templates/employee/view\_site\_details.html

{% extends 'base.html' %}

{% block content %}

<h2>Site Details</h2>

{% for location in locations %}

<h3>{{ location.Name }}</h3>

<p>{{ location.Description }}</p>

<p><strong>Coordinates:</strong> {{ location.Latitude }}, {{ location.Longitude }}</p>

<hr>

{% endfor %}

{% endblock %}

File: templates/employee/customer\_reviews.html

{% extends 'base.html' %}

{% block content %}

<h2>Customer Reviews</h2>

<table>

<tr>

<th>Feedback ID</th>

<th>Customer Trip ID</th>

<th>Rating</th>

<th>Feedback</th>

<th>Timestamp</th>

</tr>

{% for feedback in reviews %}

<tr>

<td>{{ feedback.FeedbackID }}</td>

<td>{{ feedback.CustomerTripID }}</td>

<td>{{ feedback.Rating }}</td>

<td>{{ feedback.Feedback }}</td>

<td>{{ feedback.Timestamp.strftime('%Y-%m-%d %H:%M') }}</td>

</tr>

{% endfor %}

</table>

{% endblock %}

File: templates/admin/index.html

{% extends 'base.html' %}

{% block content %}

<h2>Admin Dashboard</h2>

<ul>

<li><a href="{{ url\_for('admin.create\_employee') }}">Create Employee</a></li>

<li><a href="{{ url\_for('admin.view\_trips') }}">View Trips</a></li>

<li><a href="{{ url\_for('admin.view\_site\_details') }}">View Site Details</a></li>

<li><a href="{{ url\_for('admin.override\_assignments') }}">Override Assignments</a></li>

</ul>

{% endblock %}

File: templates/admin/create\_employee.html

{% extends 'base.html' %}

{% block content %}

<h2>Create Employee</h2>

<form method="post">

<label>First Name:</label><br>

<input type="text" name="first\_name" required><br><br>

<label>Last Name:</label><br>

<input type="text" name="last\_name" required><br><br>

<label>Username:</label><br>

<input type="text" name="username" required><br><br>

<label>Email:</label><br>

<input type="email" name="email" required><br><br>

<label>Position:</label><br>

<input type="text" name="position" required><br><br>

<label>Password:</label><br>

<input type="password" name="password" required><br><br>

<button type="submit">Create Employee</button>

</form>

{% endblock %}

File: templates/admin/view\_trips.html

{% extends 'base.html' %}

{% block content %}

<h2>View Trips</h2>

<table>

<tr>

<th>Trip ID</th>

<th>Route</th>

<th>Date</th>

<th>Total Distance</th>

</tr>

{% for trip in trips %}

<tr>

<td>{{ trip.TripID }}</td>

<td>{{ trip.route.start\_location.Name }} to {{ trip.route.end\_location.Name }}</td>

<td>{{ trip.TripDate.strftime('%Y-%m-%d %H:%M') }}</td>

<td>{{ trip.TotalDistance }}</td>

</tr>

{% endfor %}

</table>

{% endblock %}

File: templates/admin/view\_site\_details.html

{% extends 'base.html' %}

{% block content %}

<h2>Site Details</h2>

<form method="post">

<label>Select Location to Update:</label><br>

<select name="location\_id" required>

{% for location in locations %}

<option value="{{ location.LocationID }}">{{ location.Name }}</option>

{% endfor %}

</select><br><br>

<label>New Name:</label><br>

<input type="text" name="name" required><br><br>

<label>New Description:</label><br>

<textarea name="description" rows="4" cols="50" required></textarea><br><br>

<button type="submit">Update Site Details</button>

</form>

<h3>All Sites</h3>

<table>

<tr>

<th>Location ID</th>

<th>Name</th>

<th>Description</th>

<th>Coordinates</th>

</tr>

{% for location in locations %}

<tr>

<td>{{ location.LocationID }}</td>

<td>{{ location.Name }}</td>

<td>{{ location.Description }}</td>

<td>{{ location.Latitude }}, {{ location.Longitude }}</td>

</tr>

{% endfor %}

</table>

{% endblock %}

File: templates/admin/override\_assignments.html

{% extends 'base.html' %}

{% block content %}

<h2>Override Assignments</h2>

<p>This feature allows you to override emergency team assignments.</p>

<!-- Placeholder for future implementation -->

{% endblock %}

File: templates/superadmin/index.html

{% extends 'base.html' %}

{% block content %}

<h2>Super Admin Dashboard</h2>

<div class="carousel">

<a href="{{ url\_for('superadmin.create\_user') }}">Create User</a>

<a href="{{ url\_for('superadmin.manage\_users') }}">Manage Users</a>

<a href="{{ url\_for('superadmin.create\_site\_details') }}">Create Site Details</a>

<a href="{{ url\_for('superadmin.manage\_site\_details') }}">Manage Site Details</a>

</div>

{% endblock %}

File: templates/superadmin/create\_user.html

{% extends 'base.html' %}

{% block content %}

<h2>Create New User</h2>

<form method="post">

{{ form.hidden\_tag() }}

<label>First Name:</label><br>

{{ form.first\_name(size=32) }}<br><br>

<label>Last Name:</label><br>

{{ form.last\_name(size=32) }}<br><br>

<label>Username:</label><br>

{{ form.username(size=32) }}<br><br>

<label>Email:</label><br>

{{ form.email(size=32) }}<br><br>

<label>Account Type:</label><br>

{{ form.account\_type() }}<br><br>

<label>Permission Tier:</label><br>

{{ form.permission\_tier() }}<br><br>

<label>Password:</label><br>

{{ form.password(size=32) }}<br><br>

<label>Repeat Password:</label><br>

{{ form.password2(size=32) }}<br><br>

{{ form.submit() }}

</form>

{% endblock %}

File: templates/superadmin/manage\_users.html

{% extends 'base.html' %}

{% block content %}

<h2>Manage Users</h2>

<table>

<tr>

<th>Username</th>

<th>Email</th>

<th>Account Type</th>

<th>Permission Tier</th>

<th>Actions</th>

</tr>

{% for user in users %}

<tr>

<td>{{ user.Username }}</td>

<td>{{ user.Email }}</td>

<td>{{ user.AccountType }}</td>

<td>{{ user.PermissionTier }}</td>

<td>

{% if user.PermissionTier < current\_user.PermissionTier %}

<form method="post" style="display:inline;">

<input type="hidden" name="user\_id" value="{{ user.AccountID }}">

<input type="hidden" name="action" value="delete">

<button type="submit">Delete</button>

</form>

{% else %}

<span>Cannot modify</span>

{% endif %}

</td>

</tr>

{% endfor %}

</table>

<a href="{{ url\_for('superadmin.create\_user') }}">Create New User</a>

{% endblock %}

File: templates/superadmin/create\_site\_details.html

{% extends 'base.html' %}

{% block content %}

<h2>Create Site Details</h2>

<form method="post">

<label>Name:</label><br>

<input type="text" name="name" required><br><br>

<label>Description:</label><br>

<textarea name="description" rows="4" cols="50" required></textarea><br><br>

<label>Latitude:</label><br>

<input type="text" name="latitude" required><br><br>

<label>Longitude:</label><br>

<input type="text" name="longitude" required><br><br>

<button type="submit">Create Site</button>

</form>

{% endblock %}

File: templates/superadmin/manage\_site\_details.html

{% extends 'base.html' %}

{% block content %}

<h2>Manage Site Details</h2>

<form method="post">

<label>Select Location to Update:</label><br>

<select name="location\_id" required>

{% for location in locations %}

<option value="{{ location.LocationID }}">{{ location.Name }}</option>

{% endfor %}

</select><br><br>

<label>New Name:</label><br>

<input type="text" name="name" required><br><br>

<label>New Description:</label><br>

<textarea name="description" rows="4" cols="50" required></textarea><br><br>

<button type="submit">Update Site Details</button>

</form>

<h3>All Sites</h3>

<table>

<tr>

<th>Location ID</th>

<th>Name</th>

<th>Description</th>

<th>Coordinates</th>

</tr>

{% for location in locations %}

<tr>

<td>{{ location.LocationID }}</td>

<td>{{ location.Name }}</td>

<td>{{ location.Description }}</td>

<td>{{ location.Latitude }}, {{ location.Longitude }}</td>

</tr>

{% endfor %}

</table>

{% endblock %}

File: static/styles.css

body {

font-family: Arial, sans-serif;

margin: 0;

padding: 0;

}

header {

background-color: #3c8dbc;

color: white;

padding: 10px 20px;

}

header h1 {

display: inline-block;

margin: 0;

}

nav {

display: inline-block;

float: right;

}

nav a {

color: white;

margin-left: 15px;

text-decoration: none;

}

main {

padding: 20px;

}

.flashes {

list-style-type: none;

padding: 0;

}

.flashes li {

background-color: #ffdddd;

padding: 10px;

margin-bottom: 10px;

border-left: 4px solid #f44336;

}

table {

width: 100%;

border-collapse: collapse;

margin-bottom: 20px;

}

table, th, td {

border: 1px solid #dddddd;

}

th, td {

padding: 8px;

text-align: left;

}

.carousel {

display: flex;

overflow-x: auto;

margin-bottom: 20px;

}

.carousel a {

flex: 0 0 auto;

margin: 10px;

padding: 20px;

background-color: #3c8dbc;

color: white;

text-decoration: none;

text-align: center;

border-radius: 5px;

}

.carousel a:hover {

background-color: #357ca5;

}

button {

padding: 5px 10px;

margin: 2px;

background-color: #3c8dbc;

color: white;

border: none;

border-radius: 3px;

cursor: pointer;

}

button:hover {

background-color: #357ca5;

}

input[type="text"], input[type="email"], input[type="password"], select, textarea {

width: 100%;

padding: 8px;

margin: 4px 0 12px 0;

box-sizing: border-box;

}

form button, form a {

margin-top: 10px;

}

File: requirements.txt

Flask==2.0.3

Flask-Login==0.5.0

Flask-Migrate==3.1.0

Flask-SQLAlchemy==2.5.1

Flask-WTF==1.0.0

Werkzeug==2.0.3

PyMySQL==1.0.2

File: README.md

# CCC Emergency Map System

## Overview

The CCC Emergency Map System is a Flask-based web application designed to manage emergency trips, distress alerts, and user feedback. It supports multiple user roles, including Customers, Employees, Admins, and Super Admins, each with specific permissions and functionalities.

## Features

- \*\*User Authentication\*\*: Secure registration and login for all user types.

- \*\*Role-Based Access Control\*\*: Different dashboards and functionalities based on user roles.

- \*\*Trip Management\*\*: Employees and Admins can create and manage trips.

- \*\*Emergency Distress Alerts\*\*: Customers can create distress alerts, which Employees can manage.

- \*\*Site Details Management\*\*: Admins and Super Admins can view and update site details.

- \*\*User Management\*\*: Super Admins can create and manage Admin and Super Admin accounts.

- \*\*Customer Feedback\*\*: Customers can leave reviews and ratings for trips.

## Installation

### Prerequisites

- Python 3.x

- MySQL Server

### Steps

1. \*\*Clone the Repository\*\*

```bash

git clone https://github.com/yourusername/ccc\_emergency\_map.git

cd ccc\_emergency\_map

```

2. \*\*Set Up Virtual Environment\*\*

```bash

python3 -m venv venv

source venv/bin/activate # On Windows use `venv\Scripts\activate`

```

3. \*\*Install Dependencies\*\*

```bash

pip install -r requirements.txt

```

4. \*\*Configure Environment Variables\*\*

Create a `.env` file in the project root (optional) and set the following variables:

```env

SECRET\_KEY=your\_secret\_key

MYSQL\_HOST=localhost

MYSQL\_USER=root

MYSQL\_PASSWORD=your\_mysql\_password

MYSQL\_DB=ccc\_emergency\_map

```

Alternatively, set these variables in your system environment.

5. \*\*Set Up the Database\*\*

- \*\*Create Database and Tables\*\*

Execute your provided MySQL SQL script to create the `ccc\_emergency\_map` database and all tables.

```bash

mysql -u root -p your\_mysql\_password < ccc\_emergency\_map\_schema.sql

```

- \*\*Initialize Database Migrations\*\*

```bash

flask db init

flask db migrate -m "Initial migration."

flask db upgrade

```

6. \*\*Run the Application\*\*

```bash

flask run

```

The application will be accessible at `http://localhost:5000/`.

## Usage

1. \*\*Registration\*\*

- Users can register as Customers via the registration page.

- Super Admins can create Admin and Super Admin accounts through the Super Admin dashboard.

2. \*\*Login\*\*

- Navigate to the login page and enter your credentials.

- Upon successful login, you'll be redirected to your respective dashboard based on your role.

3. \*\*Dashboards\*\*

- \*\*Customer Dashboard\*\*: View trips, create distress alerts, view site details, and leave reviews.

- \*\*Employee Dashboard\*\*: Manage trips, handle assigned alerts, view all distress trips, view site details, and view customer reviews.

- \*\*Admin Dashboard\*\*: Create Employee accounts, view trips, manage site details, and override assignments.

- \*\*Super Admin Dashboard\*\*: Create and manage Admin and Super Admin accounts, and manage site details.

## Security Considerations

- \*\*Password Handling\*\*: Passwords are hashed using Werkzeug's security utilities.

- \*\*Environment Variables\*\*: Sensitive information like secret keys and database passwords should be set as environment variables and not hardcoded.

- \*\*Role-Based Access\*\*: Decorators ensure that users can only access functionalities permitted to their roles.

## Contributing

Contributions are welcome! Please fork the repository and submit a pull request for any enhancements or bug fixes.

## License

This project is licensed under the MIT License. See the [LICENSE](LICENSE) file for details.

File: ccc\_emergency\_map\_schema.sql

-- Create and use the schema if it does not exist

CREATE DATABASE IF NOT EXISTS ccc\_emergency\_map;

USE ccc\_emergency\_map;

-- Drop tables if they exist to avoid conflicts

DROP TABLE IF EXISTS ActivityLog;

DROP TABLE IF EXISTS EmergencyIncidentAnalysis;

DROP TABLE IF EXISTS RouteConditionLog;

DROP TABLE IF EXISTS EmergencyResponseTeamMember;

DROP TABLE IF EXISTS EmergencyResponseTeam;

DROP TABLE IF EXISTS EmergencyResource;

DROP TABLE IF EXISTS EmergencyDistressAssignmentQueue;

DROP TABLE IF EXISTS CustomerFeedback;

DROP TABLE IF EXISTS EmergencyDistressAlerts;

DROP TABLE IF EXISTS CustomerTrip;

DROP TABLE IF EXISTS Trip;

DROP TABLE IF EXISTS TripType;

DROP TABLE IF EXISTS Geofences;

DROP TABLE IF EXISTS PointsOfInterest;

DROP TABLE IF EXISTS Route;

DROP TABLE IF EXISTS LocationsOfInterest;

DROP TABLE IF EXISTS Employee;

DROP TABLE IF EXISTS Customer;

DROP TABLE IF EXISTS Account;

DROP TABLE IF EXISTS Permissions;

-- Permissions Table

CREATE TABLE Permissions (

PermissionID TINYINT PRIMARY KEY,

PermissionName VARCHAR(100) UNIQUE NOT NULL

);

-- Set default permissions

INSERT INTO Permissions (PermissionID, PermissionName) VALUES

(1, 'Customer'),

(2, 'Employee'),

(3, 'Admin'),

(4, 'Super Admin');

-- Account Table

CREATE TABLE Account (

AccountID INT PRIMARY KEY AUTO\_INCREMENT,

Username VARCHAR(100) UNIQUE NOT NULL,

PasswordHash VARBINARY(255) NOT NULL,

Email VARCHAR(100) UNIQUE NOT NULL,

PhoneNumber VARCHAR(15),

Address TEXT,

RegistrationDate TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

FirstName VARCHAR(100) NOT NULL,

LastName VARCHAR(100) NOT NULL,

AccountType ENUM('Customer', 'Employee', 'Admin', 'Super Admin') NOT NULL,

PermissionTier TINYINT NOT NULL DEFAULT 0, -- Default to no access

FOREIGN KEY (PermissionTier) REFERENCES Permissions(PermissionID)

);

-- Customer Table

CREATE TABLE Customer (

CustomerID INT PRIMARY KEY,

EmergencyContactName VARCHAR(100),

EmergencyContactPhone VARCHAR(15),

FOREIGN KEY (CustomerID) REFERENCES Account(AccountID)

);

-- Employee Table

CREATE TABLE Employee (

EmployeeID INT PRIMARY KEY,

Position VARCHAR(100) NOT NULL,

HireDate TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

Status VARCHAR(50) DEFAULT 'Available' NOT NULL,

RoleUpdateRequired TINYINT DEFAULT 0,

FOREIGN KEY (EmployeeID) REFERENCES Account(AccountID)

);

-- LocationsOfInterest Table

CREATE TABLE LocationsOfInterest (

LocationID INT PRIMARY KEY AUTO\_INCREMENT,

Name VARCHAR(100) NOT NULL,

Description TEXT,

Latitude DECIMAL(10, 7) NOT NULL,

Longitude DECIMAL(10, 7) NOT NULL,

Type VARCHAR(50),

Elevation DECIMAL(8, 2),

TerrainType VARCHAR(50),

AccessibilityNotes TEXT,

LastUpdated TIMESTAMP DEFAULT CURRENT\_TIMESTAMP ON UPDATE CURRENT\_TIMESTAMP

);

-- Route Table

CREATE TABLE Route (

RouteID INT PRIMARY KEY AUTO\_INCREMENT,

StartLocationID INT NOT NULL,

EndLocationID INT NOT NULL,

Distance DECIMAL(5, 2) NOT NULL,

EstimatedTime DECIMAL(5, 2),

DifficultyLevel VARCHAR(50),

AgeRequirement INT,

FOREIGN KEY (StartLocationID) REFERENCES LocationsOfInterest(LocationID),

FOREIGN KEY (EndLocationID) REFERENCES LocationsOfInterest(LocationID)

);

-- PointsOfInterest Table

CREATE TABLE PointsOfInterest (

POIID INT PRIMARY KEY AUTO\_INCREMENT,

Name VARCHAR(100) NOT NULL,

Description TEXT,

Latitude DECIMAL(10, 7) NOT NULL,

Longitude DECIMAL(10, 7) NOT NULL,

RouteID INT NOT NULL,

Type VARCHAR(50),

FOREIGN KEY (RouteID) REFERENCES Route(RouteID)

);

-- Geofences Table

CREATE TABLE Geofences (

GeofenceID INT PRIMARY KEY AUTO\_INCREMENT,

POIID INT NOT NULL,

Radius DECIMAL(6, 2) NOT NULL,

AlertType VARCHAR(50),

FOREIGN KEY (POIID) REFERENCES PointsOfInterest(POIID)

);

-- TripType Table

CREATE TABLE TripType (

TripTypeID INT PRIMARY KEY AUTO\_INCREMENT,

TypeName VARCHAR(100) UNIQUE NOT NULL,

Description TEXT

);

-- Trip Table

CREATE TABLE Trip (

TripID INT PRIMARY KEY AUTO\_INCREMENT,

RouteID INT NOT NULL,

TripTypeID INT NOT NULL,

TripDate TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

TotalDistance DECIMAL(5, 2),

FOREIGN KEY (RouteID) REFERENCES Route(RouteID),

FOREIGN KEY (TripTypeID) REFERENCES TripType(TripTypeID)

);

-- CustomerTrip Junction Table

CREATE TABLE CustomerTrip (

CustomerTripID INT PRIMARY KEY AUTO\_INCREMENT,

CustomerID INT NOT NULL,

TripID INT NOT NULL,

FOREIGN KEY (CustomerID) REFERENCES Customer(CustomerID),

FOREIGN KEY (TripID) REFERENCES Trip(TripID),

UNIQUE(CustomerID, TripID)

);

-- EmergencyDistressAlerts Table

CREATE TABLE EmergencyDistressAlerts (

AlertID INT PRIMARY KEY AUTO\_INCREMENT,

CustomerTripID INT NOT NULL,

Latitude DECIMAL(10, 7) NOT NULL,

Longitude DECIMAL(10, 7) NOT NULL,

Timestamp TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

Status VARCHAR(50) NOT NULL,

SeverityLevel ENUM('Low', 'Medium', 'High', 'Critical') NOT NULL DEFAULT 'Medium',

ResponseTime DATETIME,

ResolutionTime DATETIME,

ResolutionNotes TEXT,

AssignedEmployeeID INT,

FOREIGN KEY (CustomerTripID) REFERENCES CustomerTrip(CustomerTripID),

FOREIGN KEY (AssignedEmployeeID) REFERENCES Employee(EmployeeID)

);

-- CustomerFeedback Table

CREATE TABLE CustomerFeedback (

FeedbackID INT PRIMARY KEY AUTO\_INCREMENT,

CustomerTripID INT NOT NULL,

Feedback TEXT,

Rating INT CHECK (Rating BETWEEN 1 AND 5),

Timestamp TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

FOREIGN KEY (CustomerTripID) REFERENCES CustomerTrip(CustomerTripID)

);

-- EmergencyDistressAssignmentQueue Table

CREATE TABLE EmergencyDistressAssignmentQueue (

QueueID INT PRIMARY KEY AUTO\_INCREMENT,

AlertID INT NOT NULL,

CreatedAt TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

Processed TINYINT(1) DEFAULT 0,

FOREIGN KEY (AlertID) REFERENCES EmergencyDistressAlerts(AlertID)

);

-- EmergencyResource Table

CREATE TABLE EmergencyResource (

ResourceID INT PRIMARY KEY AUTO\_INCREMENT,

ResourceName VARCHAR(100) NOT NULL,

ResourceType ENUM('Vehicle', 'Equipment', 'Personnel') NOT NULL,

CurrentLocationID INT,

Status ENUM('Available', 'In Use', 'Out of Service') DEFAULT 'Available',

LastUpdated TIMESTAMP DEFAULT CURRENT\_TIMESTAMP ON UPDATE CURRENT\_TIMESTAMP,

FOREIGN KEY (CurrentLocationID) REFERENCES LocationsOfInterest(LocationID)

);

-- EmergencyResponseTeam Table

CREATE TABLE EmergencyResponseTeam (

TeamID INT PRIMARY KEY AUTO\_INCREMENT,

TeamName VARCHAR(100) NOT NULL,

LeadEmployeeID INT,

FOREIGN KEY (LeadEmployeeID) REFERENCES Employee(EmployeeID)

);

-- EmergencyResponseTeamMember Junction Table

CREATE TABLE EmergencyResponseTeamMember (

TeamID INT,

EmployeeID INT,

Role VARCHAR(50),

PRIMARY KEY (TeamID, EmployeeID),

FOREIGN KEY (TeamID) REFERENCES EmergencyResponseTeam(TeamID),

FOREIGN KEY (EmployeeID) REFERENCES Employee(EmployeeID)

);

-- RouteConditionLog Table

CREATE TABLE RouteConditionLog (

LogID INT PRIMARY KEY AUTO\_INCREMENT,

RouteID INT NOT NULL,

ConditionType ENUM('Normal', 'Caution', 'Danger', 'Closed') NOT NULL,

Description TEXT,

ReportedBy INT,

ReportedAt TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

ResolvedAt TIMESTAMP NULL,

FOREIGN KEY (RouteID) REFERENCES Route(RouteID),

FOREIGN KEY (ReportedBy) REFERENCES Employee(EmployeeID)

);

-- EmergencyIncidentAnalysis Table

CREATE TABLE EmergencyIncidentAnalysis (

AnalysisID INT PRIMARY KEY AUTO\_INCREMENT,

AlertID INT NOT NULL,

ResponseTime INT, -- in minutes

ResolutionTime INT, -- in minutes

SuccessFactors TEXT,

ImprovementAreas TEXT,

AnalyzedBy INT,

AnalysisDate TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

FOREIGN KEY (AlertID) REFERENCES EmergencyDistressAlerts(AlertID),

FOREIGN KEY (AnalyzedBy) REFERENCES Employee(EmployeeID)

);

-- ActivityLog Table

CREATE TABLE ActivityLog (

LogID INT PRIMARY KEY AUTO\_INCREMENT,

ActivityType VARCHAR(100),

TableName VARCHAR(100),

RecordID VARCHAR(100),

OperationType VARCHAR(10),

Description TEXT,

Username VARCHAR(100),

Timestamp TIMESTAMP DEFAULT CURRENT\_TIMESTAMP

);

-- Triggers for Activity Logging

DELIMITER $$

-- Trigger to log inserts on Account table

CREATE TRIGGER trg\_Account\_AfterInsert

AFTER INSERT ON Account

FOR EACH ROW

BEGIN

INSERT INTO ActivityLog (ActivityType, TableName, RecordID, OperationType, Description, Username)

VALUES ('Account Created', 'Account', NEW.AccountID, 'INSERT', CONCAT('Account created with Username: ', NEW.Username), 'SYSTEM');

END$$

-- Trigger to log updates on Account table

CREATE TRIGGER trg\_Account\_AfterUpdate

AFTER UPDATE ON Account

FOR EACH ROW

BEGIN

INSERT INTO ActivityLog (ActivityType, TableName, RecordID, OperationType, Description, Username)

VALUES ('Account Updated', 'Account', NEW.AccountID, 'UPDATE', CONCAT('Account updated with Username: ', NEW.Username), 'SYSTEM');

END$$

-- Trigger to log inserts on Customer table

CREATE TRIGGER trg\_Customer\_AfterInsert

AFTER INSERT ON Customer

FOR EACH ROW

BEGIN

INSERT INTO ActivityLog (ActivityType, TableName, RecordID, OperationType, Description, Username)

VALUES ('Customer Created', 'Customer', NEW.CustomerID, 'INSERT', CONCAT('Customer created with ID: ', NEW.CustomerID), 'SYSTEM');

END$$

-- Trigger to log updates on Customer table

CREATE TRIGGER trg\_Customer\_AfterUpdate

AFTER UPDATE ON Customer

FOR EACH ROW

BEGIN

INSERT INTO ActivityLog (ActivityType, TableName, RecordID, OperationType, Description, Username)

VALUES ('Customer Updated', 'Customer', NEW.CustomerID, 'UPDATE', CONCAT('Customer updated with ID: ', NEW.CustomerID), 'SYSTEM');

END$$

-- Trigger to log inserts on Employee table

CREATE TRIGGER trg\_Employee\_AfterInsert

AFTER INSERT ON Employee

FOR EACH ROW

BEGIN

INSERT INTO ActivityLog (ActivityType, TableName, RecordID, OperationType, Description, Username)

VALUES ('Employee Created', 'Employee', NEW.EmployeeID, 'INSERT', CONCAT('Employee created with ID: ', NEW.EmployeeID), 'SYSTEM');

END$$

-- Trigger to log updates on Employee table

CREATE TRIGGER trg\_Employee\_AfterUpdate

AFTER UPDATE ON Employee

FOR EACH ROW

BEGIN

INSERT INTO ActivityLog (ActivityType, TableName, RecordID, OperationType, Description, Username)

VALUES ('Employee Updated', 'Employee', NEW.EmployeeID, 'UPDATE', CONCAT('Employee updated with ID: ', NEW.EmployeeID), 'SYSTEM');

END$$

-- Trigger to log inserts on Trip table

CREATE TRIGGER trg\_Trip\_AfterInsert

AFTER INSERT ON Trip

FOR EACH ROW

BEGIN

INSERT INTO ActivityLog (ActivityType, TableName, RecordID, OperationType, Description, Username)

VALUES ('Trip Created', 'Trip', NEW.TripID, 'INSERT', CONCAT('Trip created with ID: ', NEW.TripID), 'SYSTEM');

END$$

-- Trigger to log updates on Trip table

CREATE TRIGGER trg\_Trip\_AfterUpdate

AFTER UPDATE ON Trip

FOR EACH ROW

BEGIN

INSERT INTO ActivityLog (ActivityType, TableName, RecordID, OperationType, Description, Username)

VALUES ('Trip Updated', 'Trip', NEW.TripID, 'UPDATE', CONCAT('Trip updated with ID: ', NEW.TripID), 'SYSTEM');

END$$

-- Trigger to log inserts on EmergencyDistressAlerts table

CREATE TRIGGER trg\_EmergencyDistressAlerts\_AfterInsert

AFTER INSERT ON EmergencyDistressAlerts

FOR EACH ROW

BEGIN

INSERT INTO ActivityLog (ActivityType, TableName, RecordID, OperationType, Description, Username)

VALUES ('Emergency Distress Alert Created', 'EmergencyDistressAlerts', NEW.AlertID, 'INSERT', CONCAT('Emergency distress alert created with ID: ', NEW.AlertID), 'SYSTEM');

END$$

-- Trigger to log updates on EmergencyDistressAlerts table

CREATE TRIGGER trg\_EmergencyDistressAlerts\_AfterUpdate

AFTER UPDATE ON EmergencyDistressAlerts

FOR EACH ROW

BEGIN

INSERT INTO ActivityLog (ActivityType, TableName, RecordID, OperationType, Description, Username)

VALUES ('Emergency Distress Alert Updated', 'EmergencyDistressAlerts', NEW.AlertID, 'UPDATE', CONCAT('Emergency distress alert updated with ID: ', NEW.AlertID), 'SYSTEM');

END$$

-- Trigger to log inserts on CustomerFeedback table

CREATE TRIGGER trg\_CustomerFeedback\_AfterInsert

AFTER INSERT ON CustomerFeedback

FOR EACH ROW

BEGIN

INSERT INTO ActivityLog (ActivityType, TableName, RecordID, OperationType, Description, Username)

VALUES ('Customer Feedback Created', 'CustomerFeedback', NEW.FeedbackID, 'INSERT', CONCAT('Customer feedback created with ID: ', NEW.FeedbackID), 'SYSTEM');

END$$

-- Trigger to log updates on CustomerFeedback table

CREATE TRIGGER trg\_CustomerFeedback\_AfterUpdate

AFTER UPDATE ON CustomerFeedback

FOR EACH ROW

BEGIN

INSERT INTO ActivityLog (ActivityType, TableName, RecordID, OperationType, Description, Username)

VALUES ('Customer Feedback Updated', 'CustomerFeedback', NEW.FeedbackID, 'UPDATE', CONCAT('Customer feedback updated with ID: ', NEW.FeedbackID), 'SYSTEM');

END$$

-- Trigger to log inserts on EmergencyResponseTeam table

CREATE TRIGGER trg\_EmergencyResponseTeam\_AfterInsert

AFTER INSERT ON EmergencyResponseTeam

FOR EACH ROW

BEGIN

INSERT INTO ActivityLog (ActivityType, TableName, RecordID, OperationType, Description, Username)

VALUES ('Emergency Response Team Created', 'EmergencyResponseTeam', NEW.TeamID, 'INSERT', CONCAT('Emergency response team created with ID: ', NEW.TeamID), 'SYSTEM');

END$$

-- Trigger to log updates on EmergencyResponseTeam table

CREATE TRIGGER trg\_EmergencyResponseTeam\_AfterUpdate

AFTER UPDATE ON EmergencyResponseTeam

FOR EACH ROW

BEGIN

INSERT INTO ActivityLog (ActivityType, TableName, RecordID, OperationType, Description, Username)

VALUES ('Emergency Response Team Updated', 'EmergencyResponseTeam', NEW.TeamID, 'UPDATE', CONCAT('Emergency response team updated with ID: ', NEW.TeamID), 'SYSTEM');

END$$

-- Trigger to log inserts on RouteConditionLog table

CREATE TRIGGER trg\_RouteConditionLog\_AfterInsert

AFTER INSERT ON RouteConditionLog

FOR EACH ROW

BEGIN

INSERT INTO ActivityLog (ActivityType, TableName, RecordID, OperationType, Description, Username)

VALUES ('Route Condition Log Created', 'RouteConditionLog', NEW.LogID, 'INSERT', CONCAT('Route condition log created with ID: ', NEW.LogID), 'SYSTEM');

END$$

-- Trigger to log updates on RouteConditionLog table

CREATE TRIGGER trg\_RouteConditionLog\_AfterUpdate

AFTER UPDATE ON RouteConditionLog

FOR EACH ROW

BEGIN

INSERT INTO ActivityLog (ActivityType, TableName, RecordID, OperationType, Description, Username)

VALUES ('Route Condition Log Updated', 'RouteConditionLog', NEW.LogID, 'UPDATE', CONCAT('Route condition log updated with ID: ', NEW.LogID), 'SYSTEM');

END$$

-- Trigger to log inserts on EmergencyIncidentAnalysis table

CREATE TRIGGER trg\_EmergencyIncidentAnalysis\_AfterInsert

AFTER INSERT ON EmergencyIncidentAnalysis

FOR EACH ROW

BEGIN

INSERT INTO ActivityLog (ActivityType, TableName, RecordID, OperationType, Description, Username)

VALUES ('Emergency Incident Analysis Created', 'EmergencyIncidentAnalysis', NEW.AnalysisID, 'INSERT', CONCAT('Emergency incident analysis created with ID: ', NEW.AnalysisID), 'SYSTEM');

END$$

-- Trigger to log updates on EmergencyIncidentAnalysis table

CREATE TRIGGER trg\_EmergencyIncidentAnalysis\_AfterUpdate

AFTER UPDATE ON EmergencyIncidentAnalysis

FOR EACH ROW

BEGIN

INSERT INTO ActivityLog (ActivityType, TableName, RecordID, OperationType, Description, Username)

VALUES ('Emergency Incident Analysis Updated', 'EmergencyIncidentAnalysis', NEW.AnalysisID, 'UPDATE', CONCAT('Emergency incident analysis updated with ID: ', NEW.AnalysisID), 'SYSTEM');

END$$

-- Trigger to log inserts on EmergencyResource table

CREATE TRIGGER trg\_EmergencyResource\_AfterInsert

AFTER INSERT ON EmergencyResource

FOR EACH ROW

BEGIN

INSERT INTO ActivityLog (ActivityType, TableName, RecordID, OperationType, Description, Username)

VALUES ('Emergency Resource Created', 'EmergencyResource', NEW.ResourceID, 'INSERT', CONCAT('Emergency resource created with ID: ', NEW.ResourceID), 'SYSTEM');

END$$

-- Trigger to log updates on EmergencyResource table

CREATE TRIGGER trg\_EmergencyResource\_AfterUpdate

AFTER UPDATE ON EmergencyResource

FOR EACH ROW

BEGIN

INSERT INTO ActivityLog (ActivityType, TableName, RecordID, OperationType, Description, Username)

VALUES ('Emergency Resource Updated', 'EmergencyResource', NEW.ResourceID, 'UPDATE', CONCAT('Emergency resource updated with ID: ', NEW.ResourceID), 'SYSTEM');

END$$

DELIMITER ;

-- End of Schema