Car Damage Insurance System: Project Report

# 1. Introduction

The Car Damage Insurance System is a web-based solution designed for automating the insurance claim processing in the automobile sector. The system leverages AI-powered image processing and machine learning algorithms to assess vehicle damage and calculate the claim amount based on detected damage. The goal is to streamline the process of insurance claims by automating the damage assessment, improving accuracy, and reducing human intervention.

# 2. Objective

- Automate Car Damage Assessment: The system uses AI to detect damaged car parts from an uploaded image.  
- Automate Claim Calculation: Based on the detected damage, the system calculates the estimated repair cost.  
- Ensure Claim Accuracy: The system validates the user’s insurance policy and ensures that claims adhere to policy rules.

# 3. System Overview

The system consists of two primary components:  
- Frontend: The user interface where users can interact with the system and upload car damage images.  
- Backend: The server-side logic that processes the uploaded image, handles claim calculations, stores data in the database, and manages the administrative dashboard.

## Technologies Used

- Frontend: React.js  
- Backend: Flask or Node.js + Express.js  
- AI Model: YOLO (You Only Look Once) for object detection  
- Database: MongoDB for storing user and policy data  
- OCR: Tesseract OCR for document text extraction

# 4. Workflow

## User Side Workflow

1. User Login: Users log into the system using their email and password.  
2. Policy Validation: After login, users enter their policy number. The system validates the entered policy number by checking it against the database.  
3. Image Upload: Users upload an image of their damaged car (in formats like PNG, JPG, JPEG).  
4. Damage Detection: The uploaded image is processed using the YOLO object detection model to detect damaged car parts.  
5. Claim Calculation: The system calculates the repair costs based on detected damaged parts and their quantities.  
6. Claim Submission: Once the damage is detected and the cost is calculated, users can submit the claim.

## Admin Side Workflow

1. Admin Login: Admins log into the dashboard.  
2. Claim Review: Admins can see the details of submitted claims, including:  
- The user’s policy details.  
- The detected damage parts.  
- The estimated repair costs.  
3. Decision Making: The admin manually approves or rejects the claim after reviewing the details.  
4. Email Notifications: Users are notified by email when their claim is either accepted or rejected by the admin.

# 5. System Architecture

The system architecture can be visualized in the following diagram:  
(Insert System Architecture Diagram Here)

# 6. Database Schema

## User Data (user\_info)

- email: User’s email (primary key)  
- name: Full name of the user  
- vehicle\_id: Vehicle identification number  
- car\_brand: Car's brand (e.g., Toyota, Ford)  
- model: Car's model (e.g., Corolla, Fiesta)

## Car Model Pricing (car\_models)

- brand: Car brand  
- model: Car model  
- part: Part name (e.g., bonnet, bumper)  
- price: Price of the part

# 7. Key Features

## User Features

- Upload Car Damage Image: Users can upload an image of their damaged car.  
- Claim Estimation: The system detects damaged parts and estimates repair costs.  
- Policy Validation: The system verifies the policy number entered by the user.

## Admin Features

- View Claims: Admins can view and manage claims.  
- Approve/Reject Claims: Admins can approve or reject a claim based on the details provided.

# 8. Future Enhancements

- Automated Claim Approval: Based on confidence thresholds and policy rules, future improvements could allow for automated claim approvals under certain conditions.  
- Real-time Damage Estimation: Enhance the AI model to provide more accurate and faster damage detection.  
- Mobile Application: Develop a mobile app for users to easily upload images and track their claims.

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

import uuid

from werkzeug.utils import secure\_filename

import os

from ultralytics import YOLO

import bcrypt

from collections import Counter

from dotenv import load\_dotenv

from pymongo import MongoClient

from bson.objectid import ObjectId

from datetime import datetime

import smtplib

from email.mime.text import MIMEText

from email.mime.multipart import MIMEMultipart

import openai

load\_dotenv()

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

print("SECRET\_KEY:", os.getenv('SECRET\_KEY'))

app.secret\_key = os.getenv('SECRET\_KEY')

app.config['MAIL\_SERVER'] = os.getenv('SMTP\_SERVER')

print("SMTP\_PORT:", os.getenv('SMTP\_PORT'))

app.config['MAIL\_PORT'] = int(os.getenv('SMTP\_PORT',587))

# this needs to be int!

print("SMTP\_PORT:", os.getenv('SMTP\_PORT'))

app.config['MAIL\_USE\_TLS'] =True

app.config['MAIL\_USERNAME'] = os.getenv('EMAIL\_USER')

app.config['MAIL\_PASSWORD'] = os.getenv('EMAIL\_PASSWORD')

openai.api\_key=os.getenv('OPENAI\_API\_KEY')

def connect\_to\_mongo():

    client = MongoClient('mongodb://localhost:27017/')

    db = client['car\_damage\_detection']

    return db

def send\_email(recipient, subject, message):

    sender\_email = os.getenv('EMAIL\_USER')

    sender\_password = os.getenv('EMAIL\_PASSWORD')

    try:

        print(f"Sending email to: {recipient}")  # Debug print

        print(f"Subject: {subject}")

        print(f"Message: {message}")

        msg = MIMEMultipart()

        msg['From'] = sender\_email

        msg['To'] = recipient

        msg['Subject'] = subject

        msg.attach(MIMEText(message, 'plain'))

        server = smtplib.SMTP(os.getenv('SMTP\_SERVER'), int(os.getenv('SMTP\_PORT')))

        server.starttls()

        server.login(sender\_email, sender\_password)

        server.send\_message(msg)

        server.quit()

        print(f"Email sent to {recipient}")

    except Exception as e:

        print("Error sending email:", e)

try:

    db = connect\_to\_mongo()

    print("Collections:", db.list\_collection\_names())

except Exception as e:

    print("MongoDB connection error:", e)

@app.route('/')

def home():

    return render\_template('index.html')

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

def login():

    if request.method == 'POST':

        email = request.form.get('email')

        password = request.form.get('password')

        print(f"Email : {email}")

        print(f"Password : {password}")

        if not email or not password:

            flash("Email and password are required!", "error")

            return render\_template('login.html')

        db = connect\_to\_mongo()

        user\_info = db.user\_info.find\_one({"email":email})

        if user\_info:

            stored\_password = user\_info["password"]

            if bcrypt.checkpw(password.encode('utf-8'), stored\_password.encode('utf-8')):

                session['user\_email'] = email  # Store session

                flash("Login successful!", "success")

                return redirect(url\_for('validate\_policy'))

            else:

                flash("Invalid email or password.", "error")

        else:

            flash("Database connection failed. Please try again later.", "error")

    return render\_template('login.html')

@app.route('/validate\_policy', methods=['GET', 'POST'])

def validate\_policy():

    if 'user\_email' not in session:

        flash("Please login to access this page.", "error")

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

    if request.method == 'POST':

        policy\_number = request.form.get('policy\_number')

        if not policy\_number:

            flash("Please enter a policy number.", "error")

            return redirect(url\_for('validate\_policy'))

        db = connect\_to\_mongo()

        user\_email = session.get('user\_email')

        policy = db.policy\_info.find\_one({

            "policy\_number":policy\_number,

            "email":user\_email

        })

        if policy:

            expiry\_date\_str = policy.get("expiry\_date")

            expiry\_date = datetime.strptime(expiry\_date\_str, "%Y-%m-%d")

            current\_date = datetime.now()

            if current\_date < expiry\_date:

                return render\_template("policy\_result.html", policy=policy, valid=True)

            else:

                return render\_template("policy\_result.html", policy=policy, valid=False)

        else:

            flash("Policy number not found!", "error")

            return redirect(url\_for('validate\_policy'))

    return render\_template("validate\_policy.html")

@app.route('/logout')

def logout():

    session.pop('user\_email', None)

    flash("You have been logged out.", "info")

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

model\_path = r"D:\Vehicle Damage Detection\models\model weights\best.pt"

model = YOLO(model\_path)

BASE\_DIR = os.path.dirname(os.path.abspath(\_\_file\_\_))

STATIC\_DIR = os.path.join(BASE\_DIR, 'static')

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

def dashboard():

    if 'user\_email' not in session:

        flash('Login to view dashboard', 'error')

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

    if request.method == 'POST':

        file = request.files.get('image')

        if not file:

            flash('Please upload an image.', 'error')

            return render\_template('dashboard.html')

        filename = secure\_filename(file.filename)

        if not filename.lower().endswith(('.png', '.jpg', '.jpeg')):

            flash('Invalid file type. Please upload an image.', 'error')

            return render\_template('dashboard.html')

        unique\_filename = f"{uuid.uuid4().hex}\_{filename}"

        upload\_path = os.path.join(STATIC\_DIR, unique\_filename)

        file.save(upload\_path)

        session['uploaded\_image'] = unique\_filename

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

    return render\_template('dashboard.html')

@app.route('/estimate')

def estimate():

    if 'user\_email' not in session or 'uploaded\_image' not in session:

        flash("Please upload an image first.", "error")

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

    user\_email = session['user\_email']

    user\_data = db.user\_info.find\_one({'email': user\_email})

    current\_date = datetime.now().strftime("%d-%m-%Y")

    image\_path = os.path.join(STATIC\_DIR, session['uploaded\_image'])

    result = model(image\_path)

    detected\_objects = result[0].boxes

    class\_ids = [box.cls.item() for box in detected\_objects]

    class\_counts = Counter(class\_ids)

    detected\_image\_path = os.path.join(STATIC\_DIR, 'detected\_image.jpg')

    result[0].save(detected\_image\_path)

    part\_prices = get\_part\_prices(session['user\_email'], class\_counts)

    session['latest\_part\_prices'] = part\_prices

    return render\_template('estimate.html',

                           Name=user\_data.get('name'),

                           current\_date=current\_date,

                           original\_image= session['uploaded\_image'],

                           detected\_image='detected\_image.jpg',

                           part\_prices=part\_prices,

                           parts=list(part\_prices.keys()),

                           vehicle\_id=user\_data.get('vehicle\_id', 'N/A'),

                           brand=user\_data.get('car\_brand', 'N/A'),

                           model=user\_data.get('model', 'N/A'),

                           total=sum(d['total'] for d in part\_prices.values()))

@app.route('/request\_claim', methods=['POST'])

def request\_claim():

    if 'user\_email' not in session:

        flash('Login to request claim', 'error')

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

    user\_email = session['user\_email']

    if not user\_email:

       print("User email is None")

       flash("Error: User email is not available.", "error")

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

    user\_data = db.user\_info.find\_one({'email': user\_email})

    premium\_paid = user\_data.get('premium\_paid', 0)  # Default to 0 if not found

    coverage\_type = user\_data.get('coverage\_type', 'Basic')

    send\_email(

    recipient=user\_email,

    subject="Claim Request Received",

    message=f"Hi {user\_data.get('name')},\n\nWe have received your claim request. We'll get back to you soon."

    )

    part\_prices = session.get('latest\_part\_prices', {})

    current\_date = datetime.now().strftime("%d-%m-%Y")

    if not part\_prices:

        flash('No damage detected to claim.', 'error')

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

    total\_amount = sum(p['total'] for p in part\_prices.values())

    print("Fetched from user\_info:", user\_data)

    print("Premium Paid:", premium\_paid)

    print("Coverage Type:", coverage\_type)

    db.claims.insert\_one({

        'user\_email': user\_email,

        'vehicle\_id': user\_data['vehicle\_id'],

        'policy\_number': user\_data['policy\_number'],

        'claim\_status': 'pending verification',

        'estimated\_parts': list(part\_prices.keys()),

        'total\_amount': total\_amount,

        "damage\_image": session.get('uploaded\_image'),

        'premium\_paid': premium\_paid,  # Add premium paid

        'coverage\_type': coverage\_type  # Add coverage type

    })

    flash("Claim request sent successfully!", "success")

    return render\_template('estimate.html',

                           Name=user\_data.get('name'),

                           current\_date=current\_date,

                           original\_image= session['uploaded\_image'],

                           detected\_image='detected\_image.jpg',

                           part\_prices=part\_prices,

                           parts=list(part\_prices.keys()),

                           vehicle\_id=user\_data.get('vehicle\_id', 'N/A'),

                           brand=user\_data.get('car\_brand', 'N/A'),

                           model=user\_data.get('model', 'N/A'),

                           total=sum(d['total'] for d in part\_prices.values()),

                           premium\_paid=premium\_paid,  # Add premium paid

                           coverage\_type= coverage\_type  )

@app.route('/clear\_session',methods=['GET','POST'])

def clear\_session():

    session.clear()

    flash("Logged Out Successfully.", "success")

    return redirect(url\_for('admin\_login'))

@app.route('/admin/claims')

def admin\_claims():

    print("SESSION STATE:", session)

    if not session.get('admin\_logged\_in'):

        flash("You must log in as admin to access this page.", "error")

        return redirect(url\_for('admin\_login'))

    claims = db.claims.find().sort('\_id',-1)

    return render\_template('admin\_claims.html', claims=claims)

@app.route('/admin/claim\_action/<claim\_id>/<action>')

def claim\_action(claim\_id, action):

    if not session.get('admin\_logged\_in'):

       flash("Unauthorized access", "error")

       return redirect(url\_for('admin\_login'))

    if action not in ['approved', 'rejected','runai']:

        flash('Invalid action.', 'error')

        return redirect(url\_for('admin\_claims'))

    db.claims.update\_one({'\_id': ObjectId(claim\_id)}, {'$set': {'claim\_status': action}})

    claim = db.claims.find\_one({'\_id': ObjectId(claim\_id)})

    user = db.user\_info.find\_one({'email': claim['user\_email']})

    status\_msg = "approved" if action == "approved" else "rejected"

    send\_email(

    recipient=claim['user\_email'],

    subject=f"Your Claim has been {status\_msg}",

    message=f"Hi {user.get('name')},\n\nYour claim request has been {status\_msg}.\n\nVehicle ID: {claim['vehicle\_id']}\nClaim Amount: ₹{claim['total\_amount']}\n\nThank you!"

    )

    flash(f'Claim {action} successfully.', 'success')

    return redirect(url\_for('admin\_claims'))

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

def admin\_login():

    if request.method == 'POST':

        user\_id = request.form['user\_id']

        password = request.form['password']

        print(f"User trying to login: {user\_id}")

        admin = db.admin\_users.find\_one({'user\_id': user\_id})

        print(f"Admin from DB: {admin}")

        if admin:

            print("Comparing passwords now...")

            match = bcrypt.checkpw(password.encode('utf-8'), admin['password'].encode('utf-8'))

            print(f"Password match? {match}")

        else:

            print("Admin user not found!")

        if admin and bcrypt.checkpw(password.encode('utf-8'), admin['password'].encode('utf-8')):

            session['admin\_logged\_in'] = True

            return redirect(url\_for('admin\_claims'))

        else:

            flash('Invalid admin credentials', 'error')

    return render\_template('admin\_login.html')

@app.route('/my\_claims')

def my\_claims():

    if 'user\_email' not in session:

        flash('Login to view claims', 'error')

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

    user\_email = session['user\_email']

    claims\_cursor= db.claims.find({'user\_email': user\_email})

    claims=list(claims\_cursor)

    return render\_template('my\_claims.html', claims=claims)

def get\_part\_prices(email, class\_counts):

    db = connect\_to\_mongo()

    user\_data = db.user\_info.find\_one({"email":email})

    if not user\_data:

        print("User not found")

        return {}

    car\_brand = user\_data['car\_brand']

    car\_model = user\_data['model']

    prices = {}

    for class\_id, count in class\_counts.items():

        part\_name = get\_part\_name\_from\_id(class\_id)

        if part\_name:

            price\_data = db.car\_models.find\_one({

                 "brand": {"$regex": f"^{car\_brand}$", "$options": "i"},

                 "model": {"$regex": f"^{car\_model}$", "$options": "i"},

                 "part": {"$regex": f"^{part\_name}$", "$options": "i"}

            })

            print(f"Price data : {price\_data}")

            if price\_data:

                price\_per\_part = price\_data['price']

                total\_price = price\_per\_part \* count

                prices[part\_name] = {'count': count, 'price': price\_per\_part, 'total': total\_price}

    print(f"Prices : {prices}")

    return prices

def get\_part\_name\_from\_id(class\_id):

    class\_names = ['Bonnet', 'Bumper', 'Dickey', 'Door', 'Fender', 'Light', 'Windshield']

    if 0 <= class\_id < len(class\_names):

        return class\_names[int(class\_id)]

    return None

def generate\_claim\_evaluation\_prompt(claim\_data):

    return f"""

Given the following rules and data, please evaluate the insurance claim and provide a recommendation.

### Rules:

1. The claim is valid only if the paid premium matches or exceeds the required premium for the policy type. If the premium is insufficient, the claim should be rejected.

2. The policy type determines the coverage and premium requirements.

3. The claim amount should not exceed the estimated damage repair cost. If it does, reject the claim.

4. The damage assessment must align with the policy's coverage details. If parts of the car are damaged that are not covered by the policy, reject the claim.

5. If the premium is not fully paid, reject the claim regardless of damage or coverage.

6. Provide a recommendation (approve/reject) with an explanation based on the above rules.

### Claim Data:

- Paid Premium: ₹{claim\_data['paid\_premium']}

- Policy Type: {claim\_data['policy\_type']}

- Claim Amount: ₹{claim\_data['claim\_amount']}

- Coverage Details: {', '.join(claim\_data['coverage\_details'])}

- Damage Assessment: {', '.join(claim\_data['damage\_assessment'])}

- Policy Premium Threshold: ₹{claim\_data['premium\_threshold']}

### Instructions:

- Check if the paid premium matches or exceeds the required premium.

- If insufficient, recommend rejection.

- Verify if the claim amount exceeds the estimated damage. If too high, recommend rejection.

- Ensure damages are covered by the policy. If not, recommend rejection.

- If premium is fully paid, and damage is within limits, recommend approval.

- Provide a detailed explanation for your recommendation.

"""

def evaluate\_claim\_with\_ai(claim\_data):

    prompt = generate\_claim\_evaluation\_prompt(claim\_data)

    try:

        response = openai.ChatCompletion.create(

            model="gpt-3.5-turbo",  # or "gpt-4" if you have access

            messages=[

                {"role": "system", "content": "You are an expert insurance evaluator."},

                {"role": "user", "content": prompt}

            ],

            temperature=0.3

        )

        return response['choices'][0]['message']['content']

    except Exception as e:

        print(f"Error evaluating claim: {e}")

        return "Error during AI evaluation. Please try again later."

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

    app.run(debug=True)

admin\_claim.html  
<!DOCTYPE html>

<html lang="en">

<head>

  <meta charset="UTF-8" />

  <title>Admin - Manage Claims</title>

  <meta name="viewport" content="width=device-width, initial-scale=1.0"/>

  <style>

    body {

      font-family: Arial, sans-serif;

      background: linear-gradient(135deg, #e0e7ff, #d1fae5);

      margin: 0;

      padding: 2rem;

    }

    .container {

      background-color: white;

      padding: 2rem;

      border-radius: 10px;

      box-shadow: 0 0 15px rgba(0, 0, 0, 0.1);

      max-width: 1000px;

      margin: auto;

    }

    h2 {

      text-align: center;

      color: #059669;

      margin-bottom: 1.5rem;

    }

    form.logout-form {

      display: flex;

      justify-content: flex-end;

      margin-bottom: 1rem;

    }

    button[type="submit"] {

      background-color: #059669;

      color: white;

      padding: 10px 20px;

      border: none;

      border-radius: 5px;

      font-size: 1rem;

      cursor: pointer;

    }

    button[type="submit"]:hover {

      background-color: #047857;

    }

    table {

      width: 100%;

      border-collapse: collapse;

      margin-top: 1rem;

    }

    th, td {

      padding: 10px;

      text-align: left;

      border-bottom: 1px solid #ccc;

      vertical-align: top;

    }

    th {

      background-color: #f0fdf4;

      color: #065f46;

    }

    tr:nth-child(even) {

      background-color: #f9fafb;

    }

    a {

      margin-right: 10px;

      text-decoration: none;

      color: #059669;

      font-weight: bold;

    }

    a:hover {

      text-decoration: underline;

    }

    ul {

      margin: 0;

      padding-left: 20px;

    }

  </style>

</head>

<body>

  <div class="container">

    <h2>All Claim Requests</h2>

    <form action="{{ url\_for('clear\_session') }}" method="POST" class="logout-form">

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

    </form>

    <table>

      <tr>

        <th>User Email</th>

        <th>Vehicle ID</th>

        <th>Policy Number</th>

        <th>Premium Paid</th>

        <th>Coverage Type</th>

        <th>Parts</th>

        <th>Total Amount</th>

        <th>Damage Image</th>

        <th>Action</th>

      </tr>

      {% for claim in claims %}

      <tr>

        <td>{{ claim.user\_email }}</td>

        <td>{{ claim.vehicle\_id }}</td>

        <td>{{ claim.policy\_number }}</td>

        <td>₹{{ claim.premium\_paid }}</td>

        <td>{{ claim.coverage\_type }}</td>

        <td>

          <ul>

            {% for part in claim.estimated\_parts %}

              <li>{{ part }} </li>

            {% endfor %}

          </ul>

        </td>

        <td>₹{{ claim.total\_amount }}</td>

        <td>

            {% if claim.damage\_image %}

                <img src="{{ url\_for('static', filename='detected\_image.jpg') }}" width="100" height="70" alt="Damage Image">

            {% else %}

                <span>No image</span>

            {% endif %}

        </td>

        <td>

          {% if claim.claim\_status == 'pending verification' %}

            <a href="{{ url\_for('claim\_action', claim\_id=claim.\_id, action='approved') }}">✅ Approve</a>

            <br>

            <a href="{{ url\_for('claim\_action', claim\_id=claim.\_id, action='rejected') }}">❌ Reject</a>

          {% else %}

            {{ claim.claim\_status | capitalize }}

          {% endif %}

        </td>

      </tr>

      {% endfor %}

    </table>

  </div>

</body>

</html>