**1. Directory Structure**

bash

Copy

Download

/project

│

├── .env *# Environment variables (added)*

├── app.py

├── models.py *# Added (separate model file)*

├── requirements.txt

├── static/

│ └── styles.css

└── templates/

├── index.html

├── login.html

├── register.html

└── admin.html

**2. .env**

ini

Copy

Download

SECRET\_KEY=your\_random\_secret\_key\_here

STRIPE\_SECRET\_KEY=sk\_test\_your\_stripe\_key

STRIPE\_WEBHOOK\_SECRET=whsec\_your\_webhook\_secret

DATABASE\_URI=sqlite:///db.sqlite3

**3. requirements.txt**

plaintext

Copy

Download

Flask

Flask-SQLAlchemy

stripe

python-dotenv

werkzeug

**4. models.py**

python

Copy

Download

from flask\_sqlalchemy import SQLAlchemy

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

db = SQLAlchemy()

class User(db.Model):

id = db.Column(db.Integer, primary\_key=True)

username = db.Column(db.String(80), unique=True, nullable=False)

password\_hash = db.Column(db.String(128), nullable=False) *# Hashed password*

role = db.Column(db.String(20), default='free')

subscription\_plan = db.Column(db.String(20), default='free')

feedback\_history = db.Column(db.Text, default='[]')

def set\_password(self, password):

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

def check\_password(self, password):

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

**5. app.py**

python

Copy

Download

import os

import uuid

import json

from dotenv import load\_dotenv

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

from models import db, User

import stripe

*# Load environment variables*

load\_dotenv()

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

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

app.config['SQLALCHEMY\_DATABASE\_URI'] = os.getenv('DATABASE\_URI', 'sqlite:///db.sqlite3')

db.init\_app(app)

*# Stripe setup*

stripe.api\_key = os.getenv('STRIPE\_SECRET\_KEY')

*# Create tables*

with app.app\_context():

db.create\_all()

*# Helper: Get current user*

def get\_current\_user():

if 'user\_id' in session:

return User.query.get(session['user\_id'])

return None

*# Routes*

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

def register():

if request.method == 'POST':

username = request.form['username']

password = request.form['password']

if User.query.filter\_by(username=username).first():

flash('Username already exists!', 'danger')

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

user = User(username=username, role='free')

user.set\_password(password) *# Hash password*

db.session.add(user)

db.session.commit()

flash('Registration successful! Please login.', 'success')

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

return render\_template('register.html')

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

def login():

if request.method == 'POST':

username = request.form['username']

password = request.form['password']

user = User.query.filter\_by(username=username).first()

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

session['user\_id'] = user.id

flash('Logged in successfully!', 'success')

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

flash('Invalid username or password', 'danger')

return render\_template('login.html')

@app.route('/logout')

def logout():

session.clear()

flash('Logged out successfully.', 'info')

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

@app.route('/')

@app.route('/dashboard')

def dashboard():

user = get\_current\_user()

if not user:

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

feedbacks = json.loads(user.feedback\_history)

return render\_template('index.html', user=user, feedbacks=feedbacks)

@app.route('/api/grade', methods=['POST'])

def grade():

user = get\_current\_user()

if not user:

return jsonify({'error': 'Unauthorized'}), 401

feedbacks = json.loads(user.feedback\_history)

plan\_limits = {'free': 3, 'basic': 50, 'pro': 200}

if len(feedbacks) >= plan\_limits.get(user.subscription\_plan, 3):

return jsonify({'error': 'Plan limit exceeded. Upgrade to analyze more.'}), 403

data = request.get\_json()

submission\_text = data.get('submission', '')

feedback\_text = generate\_feedback(submission\_text) *# AI logic placeholder*

feedbacks.append({

'submission': submission\_text,

'feedback': feedback\_text,

'timestamp': str(uuid.uuid4())

})

user.feedback\_history = json.dumps(feedbacks)

db.session.commit()

return jsonify({'feedback': feedback\_text})

def generate\_feedback(text):

word\_count = len(text.split())

if word\_count < 50:

return "Your submission is too short. Aim for at least 50 words."

elif "good" in text.lower():

return "Excellent work! Your analysis is clear and detailed."

else:

return "Consider adding more examples or explanations."

*# Stripe routes*

@app.route('/subscribe/<plan>')

def subscribe(plan):

user = get\_current\_user()

if not user:

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

session\_url = create\_stripe\_session(user, plan)

return redirect(session\_url)

def create\_stripe\_session(user, plan):

plan\_prices = {'free': 0, 'basic': 500, 'pro': 1000} *# cents*

checkout\_session = stripe.checkout.Session.create(

payment\_method\_types=['card'],

line\_items=[{

'price\_data': {

'currency': 'usd',

'product\_data': {'name': f'{plan.capitalize()} Plan'},

'unit\_amount': plan\_prices[plan],

},

'quantity': 1,

}],

mode='payment',

success\_url=url\_for('payment\_success', \_external=True) + f'?plan={plan}',

cancel\_url=url\_for('dashboard', \_external=True),

metadata={'user\_id': user.id, 'plan': plan}

)

return checkout\_session.url

@app.route('/payment-success')

def payment\_success():

plan = request.args.get('plan')

user = get\_current\_user()

if user:

user.subscription\_plan = plan

db.session.commit()

flash(f'Upgraded to {plan} plan!', 'success')

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

*# Admin routes*

@app.route('/admin')

def admin():

user = get\_current\_user()

if not user or user.role != 'admin':

flash('Access denied: Admins only.', 'danger')

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

users = User.query.all()

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

@app.route('/make-admin/<int:user\_id>')

def make\_admin(user\_id):

admin\_user = get\_current\_user()

if not admin\_user or admin\_user.role != 'admin':

flash('Permission denied.', 'danger')

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

user = User.query.get\_or\_404(user\_id)

user.role = 'admin'

db.session.commit()

flash(f'{user.username} is now an admin.', 'success')

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

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

app.run(debug=True)

**6. templates/index.html (Updated JavaScript)**

html

Copy

Download

Run

*<!-- Add this to <head> for flash messages -->*

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

{% if messages %}

<div class="flash-messages">

{% for category, message in messages %}

<div class="alert alert-{{ category }}">{{ message }}</div>

{% endfor %}

</div>

{% endif %}

{% endwith %}

*<!-- Updated JavaScript for error handling -->*

<script>

function analyze() {

const text = document.getElementById('submission').value.trim();

if (!text) {

alert('Please enter text to analyze.');

return;

}

fetch('/api/grade', {

method: 'POST',

headers: {'Content-Type': 'application/json'},

body: JSON.stringify({submission: text})

})

.then(res => {

if (!res.ok) throw res;

return res.json();

})

.then(data => {

document.getElementById('feedback').innerText = data.feedback;

document.getElementById('feedback-section').style.display = 'block';

})

.catch(err => {

err.json().then(e => alert(e.error || 'An error occurred.'));

});

}

</script>

**7. static/styles.css (Add Flash Messages)**

css

Copy

Download

*/\* Add to styles.css \*/*

.flash-messages {

position: fixed;

top: 20px;

right: 20px;

z-index: 1000;

}

.alert {

padding: 10px 20px;

margin-bottom: 10px;

border-radius: 5px;

}

.alert-success {

background: #d4edda;

color: #155724;

}

.alert-danger {

background: #f8d7da;

color: #721c24;

}