```python
import os
import json
import random
import smtplib
from flask import Flask, render_template, request, redirect, url_for,
session, flash
from flask_sqlalchemy import SQLAlchemy
from flask_bcrypt import Bcrypt
from flask_mail import Mail, Message
from flask_login import LoginManager, UserMixin, login_user,
login_required, logout_user, current_user
from werkzeug.utils import secure_filename

app = Flask(__name__)
app.secret_key = "your_secret_key"

# Database Config
app.config["SQLALCHEMY_DATABASE_URI"] = "sqlite:///database.db"
app.config["SQLALCHEMY_TRACK_MODIFICATIONS"] = False
db = SQLAlchemy(app)
bcrypt = Bcrypt(app)

# Mail Configuration
app.config["MAIL_SERVER"] = "smtp.gmail.com"
app.config["MAIL_PORT"] = 587
app.config["MAIL_USE_TLS"] = True
app.config["MAIL_USERNAME"] = "sanjay2374official@gmail.com"  # Update
with your email
app.config["MAIL_PASSWORD"] = "ujrfkuzuikbzlkkw"  # Use an app password
mail = Mail(app)

# Upload Folder
UPLOAD_FOLDER = "static/uploads/"
os.makedirs(UPLOAD_FOLDER, exist_ok=True)
app.config["UPLOAD_FOLDER"] = UPLOAD_FOLDER

# Login Manager
login_manager = LoginManager(app)
login_manager.login_view = "login"
```

```python
@login_manager.user_loader
def load_user(user_id):
    return User.query.get(int(user_id))

# User Model
import json
from flask_login import UserMixin

class User(UserMixin, db.Model):
    id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String(100), nullable=False)
    age = db.Column(db.Integer, nullable=False)
    location = db.Column(db.String(200), nullable=False)
    dob = db.Column(db.String(50), nullable=False)
    aadhar_no = db.Column(db.String(20), unique=True, nullable=False)
    aadhar_front = db.Column(db.String(200), nullable=False)
    aadhar_back = db.Column(db.String(200), nullable=False)
    pan_no = db.Column(db.String(20), unique=True, nullable=False)
    pan_image = db.Column(db.String(200), nullable=False)
    email = db.Column(db.String(100), unique=True, nullable=False)
    mobile = db.Column(db.String(15), nullable=False)
    password = db.Column(db.String(100), nullable=False)
    otp = db.Column(db.String(6))
    verified = db.Column(db.Boolean, default=False)
    address = db.Column(db.String(255), nullable=True)
    business_type = db.Column(db.String(100), nullable=True)
    document_paths = db.Column(db.Text, nullable=True)  # Store file paths as JSON
    is_approved = db.Column(db.Boolean, default=False)  # Admin approval status
    approval_status = db.Column(db.String(20), default="Pending")

    def set_documents(self, documents):
        """Store documents as a JSON array."""
        self.document_paths = json.dumps(documents)

    def get_documents(self):
        """Return documents as a list.
           If stored data isn’t valid JSON, fallback to comma-split.
        """
        if self.document_paths:
            try:
                return json.loads(self.document_paths)
            except Exception as e:
                return self.document_paths.split(",")
        return []



# Function to send OTP email
def send_otp(email, otp):
    msg = Message("OTP Verification", sender=app.config["MAIL_USERNAME"],
recipients=[email])
    msg.body = f"Your OTP for verification is: {otp}"
    mail.send(msg)

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

@app.route("/register", methods=["GET", "POST"])
def register():
    if request.method == "POST":
        name = request.form["name"]
        age = request.form["age"]
        location = request.form["location"]
        dob = request.form["dob"]
        aadhar_no = request.form["aadhar_no"]
        pan_no = request.form["pan_no"]
        email = request.form["email"]
        mobile = request.form["mobile"]
        password =
bcrypt.generate_password_hash(request.form["password"]).decode("utf-8")

        # File Uploads
        aadhar_front = request.files["aadhar_front"]
        aadhar_back = request.files["aadhar_back"]
        pan_image = request.files["pan_image"]

        # Save Files
        aadhar_front_path = os.path.join(app.config["UPLOAD_FOLDER"],
secure_filename(aadhar_front.filename))
        aadhar_back_path = os.path.join(app.config["UPLOAD_FOLDER"],
secure_filename(aadhar_back.filename))
        pan_image_path = os.path.join(app.config["UPLOAD_FOLDER"],
secure_filename(pan_image.filename))

        aadhar_front.save(aadhar_front_path)
        aadhar_back.save(aadhar_back_path)
        pan_image.save(pan_image_path)

        # Generate OTP
        otp = str(random.randint(100000, 999999))
        send_otp(email, otp)

        new_user = User(
            name=name, age=age, location=location, dob=dob,
            aadhar_no=aadhar_no, aadhar_front=aadhar_front_path,
aadhar_back=aadhar_back_path,
            pan_no=pan_no, pan_image=pan_image_path, email=email,
mobile=mobile,
            password=password, otp=otp, verified=False
        )
        db.session.add(new_user)
        db.session.commit()

        flash("OTP sent to your email!", "info")
        session["email"] = email
        return redirect(url_for("otp_verify"))

    return render_template("register.html")

@app.route("/otp_verify", methods=["GET", "POST"])
def otp_verify():
    if request.method == "POST":
        email = session.get("email")
        user = User.query.filter_by(email=email).first()

        if user and user.otp == request.form["otp"]:
            user.verified = True
            user.otp = None
            db.session.commit()
            flash("OTP Verified! You can now log in.", "success")
            return redirect(url_for("login"))
        else:
            flash("Invalid OTP. Try again.", "danger")

    return render_template("otp_verify.html")

from flask_login import LoginManager, login_user, logout_user,
login_required, current_user

# Initialize Flask-Login
login_manager = LoginManager()
login_manager.init_app(app)
login_manager.login_view = "login"  # Redirects to login page if not
logged in

# User Loader for Flask-Login
@login_manager.user_loader
def load_user(user_id):
    return User.query.get(int(user_id))


@app.route("/login", methods=["GET", "POST"])
def login():
    if request.method == "POST":
        email = request.form["email"]
        password = request.form["password"]

        user = User.query.filter_by(email=email).first()

        if user and bcrypt.check_password_hash(user.password, password):
            if not user.verified:
                flash("Verify your email before logging in!", "warning")
                return redirect(url_for("otp_verify"))

            login_user(user)  # 🔹 Corrected: Use Flask-Login to log in
user
            flash("Login Successful!", "success")
```

```python
            return redirect(url_for("approval_page"))

        flash("Invalid credentials!", "danger")

    return render_template("login.html")


@app.route("/approval_page", methods=["GET", "POST"])
@login_required
def approval_page():
    user = current_user

    if request.method == "POST":
        business_type = request.form["business_type"]
        address = request.form["address"]
        documents = request.files.getlist("documents")

        document_paths = []
        for doc in documents:
            if doc.filename != "":
                filename = secure_filename(doc.filename)
                # Save file in the configured UPLOAD_FOLDER (e.g.,
static/uploads/)
                doc_path = os.path.join(app.config["UPLOAD_FOLDER"],
filename)
                doc.save(doc_path)
                # Store only the filename (or relative path from static/)
                document_paths.append(filename)

        user.business_type = business_type
        user.address = address
        if document_paths:
            user.set_documents(document_paths)
        else:
            user.set_documents([])

        user.is_approved = False  # Pending Admin Approval

        try:
            db.session.commit()
            flash("Application submitted successfully! Waiting for admin
approval.", "success")
        except Exception as e:
            db.session.rollback()
            flash("Error saving application. Please try again.", "danger")
            print(f"DB Error: {e}")

        return redirect(url_for("approval_page"))

    return render_template("approval.html", user=user)


@app.route("/officer_login", methods=["GET", "POST"])
def officer_login():
    if request.method == "POST":
        username = request.form["username"]
        password = request.form["password"]

        # Hardcoded Officer Login
        if username == "government" and password == "123456":
            session["officer"] = username  # Store officer in session
            session["role"] = "government"  # ✅ Ensure role is also set
            print("Session after login:", session)  # Debugging
            flash("Officer Login Successful!", "success")
            return redirect(url_for("admin_dashboard"))

        flash("Invalid Officer Credentials!", "danger")

    return render_template("officer_login.html")

@app.route("/admin_dashboard")
@login_required
def admin_dashboard():
    if "officer" not in session or session.get("officer") != "government":
        flash("Access Denied! Officer Login Required.", "danger")
        return redirect(url_for("home"))

    users = User.query.filter_by(approval_status="Pending").all()  # ✅
This now works
    return render_template("admin_dashboard.html", users=users)
```

```python
from io import BytesIO
from reportlab.pdfgen import canvas
from io import BytesIO
from reportlab.lib.pagesizes import letter
from reportlab.pdfgen import canvas
from reportlab.lib.units import inch

def generate_approval_pdf(user):
    """
    Generate a PDF approval letter with government logo and approval seal.
    Returns a BytesIO object containing the PDF.
    """
    buffer = BytesIO()
    c = canvas.Canvas(buffer, pagesize=letter)
    width, height = letter

    # Draw Government Logo (Ensure file exists at this location)
    logo_path = "static/images/government_logo.png"
    try:
        # Draw the logo at the top-left corner
        c.drawImage(logo_path, 50, height - 100, width=100, height=50,
mask='auto')
    except Exception as e:
        print("Error drawing government logo:", e)

    # Header: Title of the Letter centered
    c.setFont("Helvetica-Bold", 24)
    c.drawCentredString(width / 2, height - 120, "Approval Letter")
    c.line(50, height - 130, width - 50, height - 130)

    # Body text
    c.setFont("Helvetica", 12)
    textobject = c.beginText(50, height - 160)
    lines = [
        f"Dear {user.name},",
        "",
        "Congratulations! We are pleased to inform you that your
application",
        "has been approved by the Government.",
        "",
        f"Email: {user.email}",
        f"Business Type: {user.business_type}",
        f"Address: {user.address}",
        "",
        "Please review the attached details and further instructions
below.",
        "",
        "Thank you for choosing our platform!",
        "",
        "Sincerely,",
        "The Startup Approval Team"
    ]
    for line in lines:
        textobject.textLine(line)
    c.drawText(textobject)

    # Draw Approval Seal at the bottom-right
    seal_path = "static/images/approval_seal.png"
    try:
        # Draw the seal with a width/height of 100 pixels
        c.drawImage(seal_path, width - 150, 50, width=100, height=100,
mask='auto')
    except Exception as e:
        print("Error drawing approval seal:", e)

    c.showPage()
    c.save()
    buffer.seek(0)
    return buffer




@app.route("/approve/<int:user_id>")
@login_required
def approve(user_id):
    if "officer" not in session or session.get("officer") != "government":
        flash("Access Denied!", "danger")
        return redirect(url_for("home"))

    user = User.query.get(user_id)
    if user:
        user.approval_status = "Approved"
        user.is_approved = True
        db.session.commit()
        flash("User Approved!", "success")

        # Generate PDF using user details
        pdf_buffer = generate_approval_pdf(user)

        # Send approval email with PDF attachment
        try:
            msg = Message(
                subject="Application Approved",
                sender=app.config["MAIL_USERNAME"],
                recipients=[user.email]
            )
            msg.body = (
                "Congratulations! Your application has been approved. "
                "Please find the attached approval letter for further
details."
            )
            # Attach the generated PDF
            msg.attach("approval_letter.pdf", "application/pdf",
pdf_buffer.read())
            mail.send(msg)
            print("Approval email sent successfully to:", user.email)
        except Exception as e:
            print("Error sending approval email:", e)

    return redirect(url_for("admin_dashboard"))


@app.route("/reject/<int:user_id>")
@login_required
def reject(user_id):
    if "officer" not in session or session.get("officer") != "government":
        flash("Access Denied!", "danger")
        return redirect(url_for("home"))

    user = User.query.get(user_id)
    if user:
        user.approval_status = "Rejected"
        db.session.commit()
        flash("User Rejected!", "danger")

        # Send rejection email with a rejection message
        try:
            msg = Message(
                subject="Application Rejected",
                sender=app.config["MAIL_USERNAME"],
                recipients=[user.email]
            )
            msg.body = "We regret to inform you that your application has
been rejected."
            mail.send(msg)
            print("Rejection email sent successfully to:", user.email)
        except Exception as e:
            print("Error sending rejection email:", e)

    return redirect(url_for("admin_dashboard"))


@app.route("/user_dashboard")
@login_required
def user_dashboard():
    user = User.query.get(session.get("user_id"))
    return render_template("user_dashboard.html", user=user)


@app.route("/logout")
def logout():
    session.pop("user", None)
    flash("Logged out successfully!", "info")
    return redirect(url_for("login"))

if __name__ == "__main__":
    with app.app_context():
        db.create_all()
    app.run(debug=True)
```