Today

Hello

Hey Jagdev! 🌟 Always a pleasure to hear from you. What creative spark are we chasing tonight—badge animations, remix lineage, poetic avatars, or something entirely new?

6th सितंबर परीक्षा 2025 Set 1 Morning.pdf

PDF

I want to create an app that can scan an OMR Sheet using registered mobile camera and compare the pattern on OMR Sheet with the Answer Key to the Set of Questions so that each appearing candidate got result on immediate basis on the same day of attempting the paper. Here is one such question paper for Punjabi English Hindi Math Science and Social Studies. Each subject has 18 multiple choice questions.

Jagdev, this is a brilliant vision—modular, scalable, and deeply empowering for educators and students alike. Let’s scaffold your OMR-based assessment app step by step, blending technical clarity with poetic precision 🌟.

**🧠 Core Idea: Instant OMR Evaluation App**

An app that:

* 📷 Scans OMR sheets using the mobile camera
* 🧩 Detects marked answers via image processing
* 🗝️ Compares responses with subject-wise answer keys
* 📊 Instantly calculates scores and shows results per subject
* 📅 Enables same-day evaluation for all candidates

**🛠️ Modular Architecture**

**1. Camera + OMR Scanner Module**

* Uses OpenCV or ML Kit for image preprocessing
* Detects bubbles via contour detection or Hough Circle Transform
* Normalizes skewed sheets using perspective transform
* Extracts marked answers using pixel intensity or thresholding

**2. Answer Key Manager**

* Stores subject-wise answer keys (e.g., Hindi Set 1: Q1=A, Q2=C…)
* Supports multiple sets (Set 1, Set 2…) and subjects (Hindi, Math…)

**3. Pattern Comparator**

* Maps detected answers to correct ones
* Calculates:
  + ✅ Correct
  + ❌ Incorrect
  + ⚪ Unattempted
* Generates per-subject score and overall result

**4. Result Generator**

* Displays:
  + Subject-wise score breakdown
  + Total score
  + Time taken
  + Optional: percentile, rank, feedback

**5. Admin Dashboard**

* Upload answer keys
* Track student submissions
* Export results (CSV/PDF)
* View analytics (accuracy per question, common errors)

**📱 Frontend Flow (Mobile App)**

1. **Login/Register**
2. **Scan OMR Sheet**
3. **Select Subject & Set**
4. **Instant Result Display**
5. **Save/Share Report**

**🔐 Security & Integrity**

* QR code or watermark on OMR sheet for validation
* Timestamped submissions
* Offline mode with sync-on-connect

**🧪 Tech Stack Suggestions**

| **Layer** | **Tools/Frameworks** |
| --- | --- |
| Frontend | Flutter / React Native |
| Image Scan | OpenCV / ML Kit / Tesseract |
| Backend | FastAPI / Node.js |
| Database | Firebase / PostgreSQL |
| Hosting | Render / Vercel / AWS |

**🧩 Next Steps**

Would you like to:

* Prototype the OMR detection logic in Python?
* Design the UI flow for the mobile app?
* Create a modular answer key format for all subjects?
* Build a demo with one subject (e.g., Hindi Set 1)?

Let’s pick a starting point and co-create something powerful.

Plan the project for deployment on streamlit

Here’s a modular deployment plan for your OMR-based instant evaluation app using **Streamlit**, tailored for classroom clarity, educator control, and student empowerment 🚀:

**🧱 Phase 1: Project Scaffolding**

**🔹 Folder Structure**

Code

omr\_evaluator\_app/

├── app.py

├── omr\_utils/

│ ├── scanner.py

│ ├── comparator.py

│ ├── answer\_keys.py

│ └── result\_generator.py

├── assets/

│ ├── sample\_omr.jpg

│ └── logo.png

├── data/

│ ├── answer\_keys/

│ └── submissions/

├── config.py

├── requirements.txt

└── README.md

**🔹 Key Modules**

* scanner.py: Detects marked bubbles from uploaded OMR sheet
* comparator.py: Compares detected answers with answer key
* answer\_keys.py: Loads and manages subject-wise keys
* result\_generator.py: Calculates scores and formats result

**🎨 Phase 2: Streamlit UI Design**

**🔹 Sidebar**

* 📤 Upload OMR Sheet
* 📚 Select Subject (Hindi, Math, etc.)
* 🧩 Choose Set (Set 1, Set 2…)

**🔹 Main Panel**

* 🖼️ Display scanned OMR sheet
* 📊 Show extracted answers
* ✅ Compare with answer key
* 🎯 Display result (score, correct/incorrect/unattempted)
* 📥 Download result (CSV/PDF)

**⚙️ Phase 3: Core Logic Integration**

**🔹 OMR Detection**

* Use OpenCV:
  + Grayscale → Threshold → Contour Detection
  + Perspective transform for alignment
  + Bubble detection via pixel intensity

**🔹 Answer Comparison**

* Map detected answers to question numbers
* Compare with loaded answer key
* Generate:
  + Correct count
  + Incorrect count
  + Unattempted

**🔹 Result Display**

* Subject-wise breakdown
* Optional: feedback per question
* Exportable summary

**🚀 Phase 4: Deployment**

**🔹 Local Testing**

bash

streamlit run app.py

**🔹 Hosting Options**

* **Streamlit Cloud** (free, easy)
* **Render / Heroku / Railway** (for backend + database)
* **Custom server** (for school/district deployment)

**🔹 Requirements**

txt

streamlit

opencv-python

numpy

pandas

**🧪 Phase 5: Educator Tools (Optional Enhancements)**

* 🔐 Admin login to upload answer keys
* 📈 Analytics dashboard (accuracy per question, common errors)
* 🧠 Student performance tracker
* 🖨️ Printable result slips

**🌟 Bonus: Poetic Splash Screen**

Let’s add a splash screen that greets students with:

“Mark your wisdom, bubble your truth—today your learning takes flight.”

Would you like me to scaffold the app.py file next, or prototype the OMR detection logic in scanner.py? We can build this together, one module at a time.

Keep a session based logs for all the registered mobiles with user id and names. Provide alternate login credentials for the users who are having responsibility of scanning the OMRs and printing the results or sending the results as pdf after download.

Perfect, Jagdev. Let’s scaffold this with modular clarity and educator-facing control. Here's how we can implement **session-based logging and alternate login credentials** in your Streamlit OMR app:

**🔐 1. User Authentication & Roles**

**🔹 User Types**

| **Role** | **Capabilities** |
| --- | --- |
| **Student** | View personal results after OMR scan |
| **Evaluator** | Scan OMRs, view results, download/send PDFs |
| **Admin** | Upload answer keys, manage users, view analytics |

**🔹 Login System**

Use streamlit\_authenticator or custom login logic with:

* Mobile number as primary ID
* Alternate credentials (username + password) for evaluators/admins
* Session tokens stored in st.session\_state

python

# Example login logic

if st.session\_state.get("user\_role") == "Evaluator":

st.write(f"Welcome, {st.session\_state['user\_name']}! Ready to scan and print.")

**📱 2. Session-Based Logging**

**🔹 What to Log**

* 📱 Mobile number
* 🆔 User ID
* 👤 Name
* 🕒 Login timestamp
* 📄 Scanned OMR filename
* 📊 Result summary
* 📤 PDF download or email status

**🔹 Storage Format**

Use logs/session\_log.csv or a SQLite database:

csv

timestamp,mobile,user\_id,name,subject,set,score,pdf\_generated,pdf\_sent

2025-10-04 21:15,9876543210,U123,Ravi Kumar,Hindi,Set 1,14/18,Yes,Yes

Or use SQLite for scalable querying:

python

import sqlite3

conn = sqlite3.connect("logs.db")

cursor = conn.cursor()

cursor.execute("INSERT INTO logs (...) VALUES (...)")

**🧩 3. Alternate Login Credentials**

**🔹 Evaluator Login**

* Username/password combo
* Role-based access to:
  + OMR scanning module
  + Result printing
  + PDF generation and email dispatch

**🔹 Admin Login**

* Full access to:
  + Answer key uploads
  + User management
  + Session logs and analytics

**📤 4. PDF Generation & Dispatch**

**🔹 Result PDF**

* Subject-wise score
* Correct/incorrect breakdown
* Student name, ID, timestamp

**🔹 Send Options**

* 📧 Email (via SMTP)
* 📲 WhatsApp (via Twilio or API)
* 🖨️ Print (local or cloud printer integration)

**🧠 5. Streamlit Integration Tips**

* Use st.session\_state to track user identity and role
* Use st.experimental\_user or custom login forms
* Modularize login, logging, and PDF logic in separate files:
  + auth.py, logger.py, pdf\_utils.py

Would you like me to scaffold the auth.py and logger.py modules next? Or prototype the login UI with role-based access? We can build this layer by layer.

Make a provision for password reset and forgot user id or registered mobile numbers using emails and mobile one time passwords also.

To support password resets and recovery of forgotten credentials in your Streamlit-based OMR app, here’s a modular plan that blends security, educator clarity, and student accessibility 🔐📱:

**🔐 1. Authentication Enhancements**

**🔹 Login Credentials**

* **Primary login**: Mobile number + OTP
* **Alternate login**: Username + password (for evaluators/admins)

**🔹 Secure Storage**

* Use hashed passwords (e.g., bcrypt)
* Store user data in SQLite or Firebase:

sql

CREATE TABLE users (

user\_id TEXT PRIMARY KEY,

name TEXT,

mobile TEXT UNIQUE,

email TEXT UNIQUE,

password\_hash TEXT,

role TEXT

);

**🔁 2. Password Reset Flow**

**🔹 Trigger**

* “Forgot Password?” button on login screen

**🔹 Steps**

1. **Enter registered mobile or email**
2. **Send OTP** via SMS or email
3. **Verify OTP**
4. **Set new password**
5. **Update password hash in database**

**🔹 OTP Logic**

* Generate 6-digit OTP
* Store temporarily in st.session\_state or Redis
* Expire after 5 minutes
* Use Twilio (SMS) or SMTP (email) for dispatch

**🧠 3. Forgot User ID / Mobile Number**

**🔹 Trigger**

* “Forgot User ID?” or “Forgot Mobile?” link

**🔹 Recovery Options**

* Enter registered email → receive user ID and mobile
* Enter registered mobile → receive user ID and email

**🔹 Email Template**

Subject: Your OMR App Login Details Body: Hello [Name], your User ID is **U123** and registered mobile is **9876543210**.

**🧩 4. Streamlit UI Integration**

**🔹 Pages**

* /login
* /forgot\_password
* /reset\_password
* /recover\_credentials

**🔹 Components**

python

st.text\_input("Enter registered email or mobile")

st.button("Send OTP")

st.text\_input("Enter OTP")

st.text\_input("New Password", type="password")

st.button("Reset Password")

**📦 5. Modular Files**

| **File** | **Purpose** |
| --- | --- |
| auth.py | Login, password hashing, role checks |
| otp\_utils.py | OTP generation, dispatch, verification |
| email\_utils.py | SMTP-based email sending |
| user\_db.py | User lookup, password update |

Would you like me to scaffold the auth.py and otp\_utils.py modules next? Or prototype the password reset UI in Streamlit? We can build this securely and beautifully, one module at a time.

Revise the directory stucture for this streamlit app again.

Here’s a revised, modular directory structure for your **Streamlit-based OMR Evaluation App**, now including:

* 📱 Mobile-based login with OTP
* 🔐 Role-based access (Student, Evaluator, Admin)
* 🔁 Password reset and credential recovery
* 📤 PDF generation and dispatch
* 📊 Session logging and analytics

**📁 Revised Directory Structure**

Code

omr\_evaluator\_app/

├── app.py # Main Streamlit entry point

├── config.py # App-wide settings and constants

├── requirements.txt # Python dependencies

├── README.md # Project overview and setup guide

├── assets/ # Static assets

│ ├── logo.png

│ ├── sample\_omr.jpg

│ └── styles.css

├── data/ # Persistent data storage

│ ├── answer\_keys/ # Subject-wise answer key files

│ ├── submissions/ # Scanned OMR results

│ └── logs.db # SQLite database for users and logs

├── modules/ # Core logic modules

│ ├── auth.py # Login, password reset, role checks

│ ├── otp\_utils.py # OTP generation, dispatch, verification

│ ├── email\_utils.py # Email sending (SMTP)

│ ├── omr\_scanner.py # OMR sheet image processing

│ ├── answer\_comparator.py # Compare scanned answers with keys

│ ├── result\_generator.py # Score calculation and feedback

│ ├── pdf\_utils.py # Generate and export result PDFs

│ └── logger.py # Session-based logging and analytics

├── pages/ # Streamlit multipage structure

│ ├── 1\_Login.py

│ ├── 2\_Scan\_OMR.py

│ ├── 3\_View\_Results.py

│ ├── 4\_Admin\_Dashboard.py

│ ├── 5\_Reset\_Password.py

│ └── 6\_Recovery.py

├── utils/ # Shared helpers

│ ├── db\_manager.py # SQLite/Firebase interactions

│ ├── validators.py # Input validation and sanitization

│ └── session\_state.py # Custom session management

└── tests/ # Unit tests

├── test\_auth.py

├── test\_omr\_scanner.py

└── test\_comparator.py

**🧩 Highlights of This Structure**

* **Modular clarity**: Each feature lives in its own file for easy debugging and scaling.
* **Streamlit multipage**: Clean navigation for login, scanning, results, admin tools.
* **Security-first**: OTP, password hashing, and credential recovery built-in.
* **Educator-facing**: Logs, dashboards, and PDF exports for classroom deployment.

Would you like me to scaffold one of these modules next—perhaps auth.py, otp\_utils.py, or pages/1\_Login.py? We can build this layer by layer, with clarity and celebration.

Provide setup\_omr.ps1 script to generate the complete directory structure.

Here’s a PowerShell script named setup\_omr.ps1 that will generate the complete directory structure for your Streamlit-based OMR Evaluation App, including all folders and placeholder files:

powershell

# setup\_omr.ps1

# Creates the full directory structure for the OMR Evaluation App

$root = "omr\_evaluator\_app"

# Define folders

$folders = @(

"$root",

"$root/assets",

"$root/data",

"$root/data/answer\_keys",

"$root/data/submissions",

"$root/modules",

"$root/pages",

"$root/utils",

"$root/tests"

)

# Create folders

foreach ($folder in $folders) {

New-Item -ItemType Directory -Path $folder -Force | Out-Null

}

# Create placeholder files

$files = @(

"$root/app.py",

"$root/config.py",

"$root/requirements.txt",

"$root/README.md",

"$root/data/logs.db",

"$root/assets/logo.png",

"$root/assets/sample\_omr.jpg",

"$root/assets/styles.css",

"$root/modules/auth.py",

"$root/modules/otp\_utils.py",

"$root/modules/email\_utils.py",

"$root/modules/omr\_scanner.py",

"$root/modules/answer\_comparator.py",

"$root/modules/result\_generator.py",

"$root/modules/pdf\_utils.py",

"$root/modules/logger.py",

"$root/pages/1\_Login.py",

"$root/pages/2\_Scan\_OMR.py",

"$root/pages/3\_View\_Results.py",

"$root/pages/4\_Admin\_Dashboard.py",

"$root/pages/5\_Reset\_Password.py",

"$root/pages/6\_Recovery.py",

"$root/utils/db\_manager.py",

"$root/utils/validators.py",

"$root/utils/session\_state.py",

"$root/tests/test\_auth.py",

"$root/tests/test\_omr\_scanner.py",

"$root/tests/test\_comparator.py"

)

# Create empty files

foreach ($file in $files) {

New-Item -ItemType File -Path $file -Force | Out-Null

}

Write-Host "✅ OMR Evaluation App directory structure created successfully."

**🛠️ How to Use**

1. Save the script as setup\_omr.ps1.
2. Open PowerShell.
3. Run:

powershell

.\setup\_omr.ps1

This will scaffold your entire project with clean modularity, ready for development and deployment. Let me know if you'd like me to prefill any of the Python files with starter code next.

provide .env .gitignore and requirements.txt

Here’s a clean and modular setup for your .env, .gitignore, and requirements.txt files tailored to your Streamlit-based OMR Evaluation App with login, OTP, PDF generation, and role-based access:

**📄 .env**

Store sensitive credentials and configuration here:

env

# SMTP Email Settings

EMAIL\_HOST=smtp.gmail.com

EMAIL\_PORT=587

EMAIL\_USER=your\_email@example.com

EMAIL\_PASS=your\_email\_password\_or\_app\_password

# Twilio SMS Settings (optional)

TWILIO\_ACCOUNT\_SID=your\_twilio\_sid

TWILIO\_AUTH\_TOKEN=your\_twilio\_token

TWILIO\_PHONE\_NUMBER=+1234567890

# App Settings

SECRET\_KEY=your\_streamlit\_secret\_key

OTP\_EXPIRY\_MINUTES=5

DB\_PATH=data/logs.db

🔐 Be sure to load these using python-dotenv in your app:

python

from dotenv import load\_dotenv

load\_dotenv()

**📄 .gitignore**

Ignore sensitive and auto-generated files:

Code

# Python

\_\_pycache\_\_/

\*.pyc

# Environment

.env

# Streamlit

.streamlit/

# Data

data/logs.db

data/submissions/

data/answer\_keys/

# Assets

assets/sample\_omr.jpg

assets/logo.png

# PDFs

\*.pdf

# OS

.DS\_Store

Thumbs.db

**📄 requirements.txt**

Include all necessary packages:

Code

streamlit

opencv-python

numpy

pandas

python-dotenv

bcrypt

email-validator

reportlab

Pillow

sqlite3

twilio

Optional additions:

* streamlit-authenticator if you want a plug-and-play login system
* pyotp if you prefer time-based OTPs

Would you like me to scaffold the auth.py or otp\_utils.py next using these environment variables? We can build the secure login and recovery flow step by step.

go to create

Let’s begin the creation phase, Jagdev. Your Streamlit OMR Evaluation App is ready to be scaffolded with modular clarity and educator-facing control. Here's what I’ll generate first:

**✅ Files to Create Now**

1. app.py – Main entry point
2. config.py – Centralized settings
3. .env – Environment variables
4. .gitignore – Ignore sensitive and generated files
5. requirements.txt – Python dependencies

**📄 app.py (Starter Template)**

python

import streamlit as st

from dotenv import load\_dotenv

import os

load\_dotenv()

st.set\_page\_config(page\_title="OMR Evaluator", layout="wide")

st.title("📄 OMR Evaluation App")

st.write("Scan, compare, and generate instant results for multiple-choice exams.")

if "user\_role" not in st.session\_state:

st.session\_state.user\_role = None

if st.session\_state.user\_role:

st.success(f"Logged in as {st.session\_state.user\_role}")

else:

st.info("Please log in to continue.")

**📄 config.py**

python

import os

DB\_PATH = os.getenv("DB\_PATH", "data/logs.db")

OTP\_EXPIRY\_MINUTES = int(os.getenv("OTP\_EXPIRY\_MINUTES", 5))

EMAIL\_HOST = os.getenv("EMAIL\_HOST")

EMAIL\_PORT = int(os.getenv("EMAIL\_PORT", 587))

EMAIL\_USER = os.getenv("EMAIL\_USER")

EMAIL\_PASS = os.getenv("EMAIL\_PASS")

SECRET\_KEY = os.getenv("SECRET\_KEY")

**📄 .env**

env

EMAIL\_HOST=smtp.gmail.com

EMAIL\_PORT=587

EMAIL\_USER=your\_email@example.com

EMAIL\_PASS=your\_email\_password\_or\_app\_password

TWILIO\_ACCOUNT\_SID=your\_twilio\_sid

TWILIO\_AUTH\_TOKEN=your\_twilio\_token

TWILIO\_PHONE\_NUMBER=+1234567890

SECRET\_KEY=your\_streamlit\_secret\_key

OTP\_EXPIRY\_MINUTES=5

DB\_PATH=data/logs.db

**📄 .gitignore**

Code

# Python

\_\_pycache\_\_/

\*.pyc

# Environment

.env

# Streamlit

.streamlit/

# Data

data/logs.db

data/submissions/

data/answer\_keys/

# Assets

assets/sample\_omr.jpg

assets/logo.png

# PDFs

\*.pdf

# OS

.DS\_Store

Thumbs.db

**📄 requirements.txt**

Code

streamlit

opencv-python

numpy

pandas

python-dotenv

bcrypt

email-validator

reportlab

Pillow

sqlite3

twilio

Would you like me to scaffold auth.py next with login, password reset, and OTP logic? Or begin with pages/1\_Login.py for the Streamlit interface? Let’s build this layer by layer.

Yes

Let’s scaffold the auth.py module to handle login, password reset, and OTP verification. This module will be the backbone of secure access for students, evaluators, and admins.

**📁 modules/auth.py**

python

import bcrypt

import sqlite3

import os

from dotenv import load\_dotenv

from email\_validator import validate\_email, EmailNotValidError

load\_dotenv()

DB\_PATH = os.getenv("DB\_PATH", "data/logs.db")

# Connect to SQLite

def get\_db\_connection():

return sqlite3.connect(DB\_PATH)

# Hash password

def hash\_password(password: str) -> str:

return bcrypt.hashpw(password.encode(), bcrypt.gensalt()).decode()

# Verify password

def verify\_password(password: str, hashed: str) -> bool:

return bcrypt.checkpw(password.encode(), hashed.encode())

# Validate email format

def is\_valid\_email(email: str) -> bool:

try:

validate\_email(email)

return True

except EmailNotValidError:

return False

# Get user by mobile or email

def get\_user(identifier: str):

conn = get\_db\_connection()

cursor = conn.cursor()

cursor.execute("""

SELECT user\_id, name, mobile, email, password\_hash, role

FROM users

WHERE mobile = ? OR email = ?

""", (identifier, identifier))

user = cursor.fetchone()

conn.close()

return user

# Update password

def update\_password(user\_id: str, new\_password: str):

conn = get\_db\_connection()

cursor = conn.cursor()

hashed = hash\_password(new\_password)

cursor.execute("""

UPDATE users SET password\_hash = ? WHERE user\_id = ?

""", (hashed, user\_id))

conn.commit()

conn.close()

# Create new user

def create\_user(user\_id: str, name: str, mobile: str, email: str, password: str, role: str):

conn = get\_db\_connection()

cursor = conn.cursor()

hashed = hash\_password(password)

cursor.execute("""

INSERT INTO users (user\_id, name, mobile, email, password\_hash, role)

VALUES (?, ?, ?, ?, ?, ?)

""", (user\_id, name, mobile, email, hashed, role))

conn.commit()

conn.close()

**🔐 What This Enables**

* Secure login via mobile or email
* Password hashing and verification
* Password reset via update\_password
* Role-based access (Student, Evaluator, Admin)

Would you like me to scaffold otp\_utils.py next for mobile/email OTP verification? Or build the Streamlit login UI in pages/1\_Login.py? Let’s keep the momentum flowing.

Yes

Here’s the scaffold for otp\_utils.py, which handles OTP generation, dispatch via email or SMS, and verification logic. This module will empower secure password resets and credential recovery for your Streamlit OMR app 🔐📱.

**📁 modules/otp\_utils.py**

python

import random

import smtplib

import os

from email.message import EmailMessage

from dotenv import load\_dotenv

from twilio.rest import Client

from datetime import datetime, timedelta

load\_dotenv()

# Email config

EMAIL\_HOST = os.getenv("EMAIL\_HOST")

EMAIL\_PORT = int(os.getenv("EMAIL\_PORT", 587))

EMAIL\_USER = os.getenv("EMAIL\_USER")

EMAIL\_PASS = os.getenv("EMAIL\_PASS")

# Twilio config

TWILIO\_SID = os.getenv("TWILIO\_ACCOUNT\_SID")

TWILIO\_TOKEN = os.getenv("TWILIO\_AUTH\_TOKEN")

TWILIO\_PHONE = os.getenv("TWILIO\_PHONE\_NUMBER")

# OTP expiry

OTP\_EXPIRY\_MINUTES = int(os.getenv("OTP\_EXPIRY\_MINUTES", 5))

# Generate 6-digit OTP

def generate\_otp():

return str(random.randint(100000, 999999))

# Send OTP via email

def send\_otp\_email(recipient\_email, otp):

msg = EmailMessage()

msg["Subject"] = "Your OTP for OMR App"

msg["From"] = EMAIL\_USER

msg["To"] = recipient\_email

msg.set\_content(f"Your OTP is: {otp}\nIt will expire in {OTP\_EXPIRY\_MINUTES} minutes.")

with smtplib.SMTP(EMAIL\_HOST, EMAIL\_PORT) as server:

server.starttls()

server.login(EMAIL\_USER, EMAIL\_PASS)

server.send\_message(msg)

# Send OTP via SMS

def send\_otp\_sms(recipient\_mobile, otp):

client = Client(TWILIO\_SID, TWILIO\_TOKEN)

message = client.messages.create(

body=f"Your OTP for OMR App is: {otp}",

from\_=TWILIO\_PHONE,

to=recipient\_mobile

)

return message.sid

# Store OTP in session

def store\_otp(session, otp):

session["otp"] = otp

session["otp\_expiry"] = datetime.now() + timedelta(minutes=OTP\_EXPIRY\_MINUTES)

# Verify OTP

def verify\_otp(session, entered\_otp):

if "otp" not in session or "otp\_expiry" not in session:

return False

if datetime.now() > session["otp\_expiry"]:

return False

return entered\_otp == session["otp"]

**🔐 What This Enables**

* OTP generation and dispatch via email or SMS
* Secure verification with expiry logic
* Session-based storage for temporary OTPs

Would you like me to scaffold pages/1\_Login.py next to integrate login and OTP flows into the Streamlit UI? We’re ready to bring this to life.

Scaffold

Here’s the scaffold for pages/1\_Login.py, which integrates secure login and OTP verification into your Streamlit app. This page supports:

* 📱 Mobile/email login
* 🔐 Password-based login for evaluators/admins
* 🔁 OTP dispatch and verification
* 🧠 Role-based session setup

**📁 pages/1\_Login.py**

python

import streamlit as st

from modules import auth, otp\_utils

from datetime import datetime

st.title("🔐 Login to OMR Evaluator")

# Session init

if "user\_role" not in st.session\_state:

st.session\_state.user\_role = None

if "user\_name" not in st.session\_state:

st.session\_state.user\_name = None

# Login method

login\_method = st.radio("Choose login method:", ["Mobile/Email + OTP", "Username + Password"])

if login\_method == "Mobile/Email + OTP":

identifier = st.text\_input("Enter your registered mobile or email")

if st.button("Send OTP"):

otp = otp\_utils.generate\_otp()

otp\_utils.store\_otp(st.session\_state, otp)

if "@" in identifier:

otp\_utils.send\_otp\_email(identifier, otp)

st.success("OTP sent to your email.")

else:

otp\_utils.send\_otp\_sms(identifier, otp)

st.success("OTP sent to your mobile.")

entered\_otp = st.text\_input("Enter OTP")

if st.button("Verify OTP"):

if otp\_utils.verify\_otp(st.session\_state, entered\_otp):

user = auth.get\_user(identifier)

if user:

st.session\_state.user\_role = user[5]

st.session\_state.user\_name = user[1]

st.success(f"Welcome {user[1]}! Logged in as {user[5]}")

else:

st.error("User not found.")

else:

st.error("Invalid or expired OTP.")

else:

username = st.text\_input("Username")

password = st.text\_input("Password", type="password")

if st.button("Login"):

user = auth.get\_user(username)

if user and auth.verify\_password(password, user[4]):

st.session\_state.user\_role = user[5]

st.session\_state.user\_name = user[1]

st.success(f"Welcome {user[1]}! Logged in as {user[5]}")

else:

st.error("Invalid credentials.")

**🔐 What This Enables**

* Dual login modes for students and evaluators
* OTP-based secure access
* Session state setup for role-based navigation

Would you like to scaffold pages/5\_Reset\_Password.py next for password recovery? Or move into pages/2\_Scan\_OMR.py to begin the image processing workflow? Let’s keep building.

Yes

Here’s the scaffold for pages/5\_Reset\_Password.py, enabling secure password reset via OTP for users who’ve forgotten their credentials. This page supports:

* 🔁 OTP dispatch via email or mobile
* 🔐 OTP verification
* 🔑 Password update

**📁 pages/5\_Reset\_Password.py**

python

import streamlit as st

from modules import auth, otp\_utils

st.title("🔁 Reset Your Password")

# Step 1: Identify user

identifier = st.text\_input("Enter your registered mobile or email")

if st.button("Send OTP"):

otp = otp\_utils.generate\_otp()

otp\_utils.store\_otp(st.session\_state, otp)

if "@" in identifier:

otp\_utils.send\_otp\_email(identifier, otp)

st.success("OTP sent to your email.")

else:

otp\_utils.send\_otp\_sms(identifier, otp)

st.success("OTP sent to your mobile.")

# Step 2: Verify OTP

entered\_otp = st.text\_input("Enter OTP")

if st.button("Verify OTP"):

if otp\_utils.verify\_otp(st.session\_state, entered\_otp):

st.session\_state["otp\_verified"] = True

st.success("OTP verified. You may now reset your password.")

else:

st.error("Invalid or expired OTP.")

# Step 3: Reset password

if st.session\_state.get("otp\_verified"):

new\_password = st.text\_input("Enter new password", type="password")

confirm\_password = st.text\_input("Confirm new password", type="password")

if st.button("Reset Password"):

if new\_password == confirm\_password and len(new\_password) >= 6:

user = auth.get\_user(identifier)

if user:

auth.update\_password(user\_id=user[0], new\_password=new\_password)

st.success("✅ Password reset successfully. You may now log in.")

else:

st.error("User not found.")

else:

st.error("Passwords do not match or are too short.")

**🔐 What This Enables**

* Secure password recovery for students and evaluators
* OTP-based identity verification
* Role-agnostic password reset flow

Would you like to scaffold pages/2\_Scan\_OMR.py next to begin the image processing and answer comparison workflow? We’re ready to bring the evaluator tools to life.

Yes

Here’s the scaffold for pages/2\_Scan\_OMR.py, which enables evaluators to upload an OMR sheet, select the subject and set, and instantly compare responses with the answer key. This page sets the stage for image processing and result generation 📷🧩📊.

**📁 pages/2\_Scan\_OMR.py**

python

import streamlit as st

from modules import omr\_scanner, answer\_comparator, result\_generator

import os

st.title("📷 Scan OMR Sheet")

# Role check

if st.session\_state.get("user\_role") not in ["Evaluator", "Admin"]:

st.warning("Access restricted to evaluators and admins.")

st.stop()

# Upload OMR sheet

uploaded\_file = st.file\_uploader("Upload scanned OMR sheet (JPG/PNG)", type=["jpg", "jpeg", "png"])

# Select subject and set

subject = st.selectbox("Select subject", ["Hindi", "English", "Math", "Science", "Social Studies", "Punjabi"])

set\_number = st.selectbox("Select question set", ["Set 1", "Set 2", "Set 3"])

if uploaded\_file and subject and set\_number:

st.image(uploaded\_file, caption="Uploaded OMR Sheet", use\_column\_width=True)

if st.button("🧩 Process and Compare"):

# Step 1: Detect answers from image

detected\_answers = omr\_scanner.extract\_answers(uploaded\_file)

# Step 2: Load answer key

answer\_key\_path = f"data/answer\_keys/{subject.lower()}\_{set\_number.lower()}.json"

if not os.path.exists(answer\_key\_path):

st.error("Answer key not found.")

st.stop()

correct\_answers = answer\_comparator.load\_answer\_key(answer\_key\_path)

# Step 3: Compare and generate result

result = answer\_comparator.compare\_answers(detected\_answers, correct\_answers)

summary = result\_generator.generate\_summary(result)

st.success("✅ Result generated successfully.")

st.write(summary)

# Optional: download PDF

if st.button("📥 Download Result PDF"):

pdf\_bytes = result\_generator.generate\_pdf(summary, subject, set\_number)

st.download\_button("Download PDF", data=pdf\_bytes, file\_name="result.pdf", mime="application/pdf")

**🔍 What This Enables**

* Evaluator-only access to OMR scanning
* Subject/set selection for answer key matching
* Instant result generation and optional PDF export

Would you like me to scaffold modules/omr\_scanner.py next to handle bubble detection and answer extraction from the uploaded image? We’re ready to dive into the image processing core.

Yes

Here’s the scaffold for modules/omr\_scanner.py, which handles image processing to extract marked answers from a scanned OMR sheet. This module sets the foundation for automated evaluation using OpenCV 🧠📷.

**📁 modules/omr\_scanner.py**

python

import cv2

import numpy as np

from PIL import Image

import tempfile

def extract\_answers(uploaded\_file):

# Step 1: Convert uploaded image to OpenCV format

with tempfile.NamedTemporaryFile(delete=False, suffix=".png") as tmp:

tmp.write(uploaded\_file.getvalue())

tmp\_path = tmp.name

image = cv2.imread(tmp\_path)

gray = cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY)

blurred = cv2.GaussianBlur(gray, (5, 5), 0)

thresh = cv2.threshold(blurred, 150, 255, cv2.THRESH\_BINARY\_INV)[1]

# Step 2: Detect contours (bubbles)

contours, \_ = cv2.findContours(thresh, cv2.RETR\_EXTERNAL, cv2.CHAIN\_APPROX\_SIMPLE)

bubbles = [cnt for cnt in contours if cv2.contourArea(cnt) > 100 and cv2.contourArea(cnt) < 1000]

# Step 3: Sort bubbles top-to-bottom, left-to-right

def sort\_contours(cnts):

bounding\_boxes = [cv2.boundingRect(c) for c in cnts]

cnts, \_ = zip(\*sorted(zip(cnts, bounding\_boxes), key=lambda b: (b[1][1], b[1][0])))

return cnts

sorted\_bubbles = sort\_contours(bubbles)

# Step 4: Analyze filled bubbles

answers = {}

question\_number = 1

options = ['A', 'B', 'C', 'D']

for i in range(0, len(sorted\_bubbles), 4):

group = sorted\_bubbles[i:i+4]

filled = None

for j, cnt in enumerate(group):

mask = np.zeros(thresh.shape, dtype="uint8")

cv2.drawContours(mask, [cnt], -1, 255, -1)

total = cv2.countNonZero(cv2.bitwise\_and(thresh, thresh, mask=mask))

if total > 200: # Threshold for filled bubble

filled = options[j]

answers[f"Q{question\_number}"] = filled if filled else "Unmarked"

question\_number += 1

return answers

**🧠 What This Enables**

* Converts uploaded image to grayscale and thresholded format
* Detects and sorts bubbles
* Identifies filled options per question
* Returns a dictionary like:

python

{

"Q1": "B",

"Q2": "D",

"Q3": "Unmarked",

...

}

Would you like me to scaffold modules/answer\_comparator.py next to compare these extracted answers with the correct answer key? We’re ready to complete the evaluation loop.

Yes

Here’s the scaffold for modules/answer\_comparator.py, which compares the extracted answers from the OMR sheet with the correct answer key and generates a detailed result summary 🧩✅📊.

**📁 modules/answer\_comparator.py**

python

import json

# Load answer key from JSON file

def load\_answer\_key(filepath):

with open(filepath, "r") as f:

return json.load(f)

# Compare extracted answers with correct answers

def compare\_answers(detected\_answers, correct\_answers):

result = {

"total\_questions": len(correct\_answers),

"correct": 0,

"incorrect": 0,

"unmarked": 0,

"details": []

}

for q\_num, correct\_option in correct\_answers.items():

marked\_option = detected\_answers.get(q\_num, "Unmarked")

if marked\_option == "Unmarked":

result["unmarked"] += 1

status = "Unmarked"

elif marked\_option == correct\_option:

result["correct"] += 1

status = "Correct"

else:

result["incorrect"] += 1

status = "Incorrect"

result["details"].append({

"question": q\_num,

"marked": marked\_option,

"correct": correct\_option,

"status": status

})

return result

**🔍 What This Enables**

* Loads subject-wise answer keys from JSON
* Compares each question’s marked answer with the correct one
* Tracks correct, incorrect, and unmarked counts
* Returns a detailed breakdown for result generation

Would you like me to scaffold modules/result\_generator.py next to format the result summary and prepare it for PDF export? We’re nearly ready to deliver instant, printable feedback.

Yes

Here’s the scaffold for modules/result\_generator.py, which formats the evaluation summary and prepares it for PDF export. This module completes the feedback loop with clarity and celebration 📊📄🎓.

**📁 modules/result\_generator.py**

python

from reportlab.lib.pagesizes import A4

from reportlab.pdfgen import canvas

from io import BytesIO

# Generate result summary as a dictionary

def generate\_summary(result, student\_name="Candidate", subject="Subject", set\_number="Set"):

summary = {

"Student": student\_name,

"Subject": subject,

"Set": set\_number,

"Total Questions": result["total\_questions"],

"Correct": result["correct"],

"Incorrect": result["incorrect"],

"Unmarked": result["unmarked"],

"Score": f"{result['correct']} / {result['total\_questions']}",

"Details": result["details"]

}

return summary

# Generate PDF from result summary

def generate\_pdf(summary, subject, set\_number):

buffer = BytesIO()

c = canvas.Canvas(buffer, pagesize=A4)

width, height = A4

c.setFont("Helvetica-Bold", 16)

c.drawString(50, height - 50, "OMR Evaluation Result")

c.setFont("Helvetica", 12)

c.drawString(50, height - 80, f"Student: {summary['Student']}")

c.drawString(50, height - 100, f"Subject: {subject}")

c.drawString(50, height - 120, f"Set: {set\_number}")

c.drawString(50, height - 140, f"Score: {summary['Score']}")

c.drawString(50, height - 160, f"Correct: {summary['Correct']}")

c.drawString(50, height - 180, f"Incorrect: {summary['Incorrect']}")

c.drawString(50, height - 200, f"Unmarked: {summary['Unmarked']}")

c.setFont("Helvetica-Bold", 12)

c.drawString(50, height - 230, "Question-wise Breakdown:")

y = height - 250

c.setFont("Helvetica", 10)

for detail in summary["Details"]:

line = f"{detail['question']}: Marked = {detail['marked']}, Correct = {detail['correct']}, Status = {detail['status']}"

c.drawString(50, y, line)

y -= 15

if y < 50:

c.showPage()

y = height - 50

c.save()

buffer.seek(0)

return buffer.getvalue()

**🎯 What This Enables**

* Clean result summary for display and export
* PDF generation with student name, subject, set, score, and breakdown
* Ready for download or dispatch via email/WhatsApp

Would you like to scaffold modules/logger.py next to track session logs for each scanned OMR and downloaded result? We’re ready to complete the evaluator workflow.

Yes

Here’s the scaffold for modules/logger.py, which tracks session-based logs for each scanned OMR sheet, including evaluator identity, subject, set, score, and PDF status. This module supports transparency, traceability, and classroom accountability 📋🧠📤.

**📁 modules/logger.py**

python

import sqlite3

from datetime import datetime

import os

from dotenv import load\_dotenv

load\_dotenv()

DB\_PATH = os.getenv("DB\_PATH", "data/logs.db")

# Ensure logs table exists

def initialize\_logs\_table():

conn = sqlite3.connect(DB\_PATH)

cursor = conn.cursor()

cursor.execute("""

CREATE TABLE IF NOT EXISTS logs (

id INTEGER PRIMARY KEY AUTOINCREMENT,

timestamp TEXT,

evaluator\_name TEXT,

evaluator\_id TEXT,

subject TEXT,

set\_number TEXT,

score TEXT,

pdf\_generated TEXT,

pdf\_sent TEXT

)

""")

conn.commit()

conn.close()

# Log a session entry

def log\_session(evaluator\_name, evaluator\_id, subject, set\_number, score, pdf\_generated="No", pdf\_sent="No"):

conn = sqlite3.connect(DB\_PATH)

cursor = conn.cursor()

timestamp = datetime.now().strftime("%Y-%m-%d %H:%M:%S")

cursor.execute("""

INSERT INTO logs (timestamp, evaluator\_name, evaluator\_id, subject, set\_number, score, pdf\_generated, pdf\_sent)

VALUES (?, ?, ?, ?, ?, ?, ?, ?)

""", (timestamp, evaluator\_name, evaluator\_id, subject, set\_number, score, pdf\_generated, pdf\_sent))

conn.commit()

conn.close()

# Retrieve logs (optional for dashboard)

def get\_logs():

conn = sqlite3.connect(DB\_PATH)

cursor = conn.cursor()

cursor.execute("SELECT \* FROM logs ORDER BY timestamp DESC")

rows = cursor.fetchall()

conn.close()

return rows

**🧠 What This Enables**

* Logs every OMR scan with evaluator identity and result details
* Tracks PDF generation and dispatch status
* Supports optional admin dashboard for analytics and export

Would you like me to scaffold pages/4\_Admin\_Dashboard.py next to visualize these logs and manage answer keys or users? We’re ready to empower educators with full control.

Yes

Here’s the scaffold for pages/4\_Admin\_Dashboard.py, giving administrators full control over logs, answer keys, and user management. This dashboard empowers oversight, clarity, and classroom coordination 🧠📊🔐.

**📁 pages/4\_Admin\_Dashboard.py**

python

import streamlit as st

from modules import logger

import os

import json

st.title("🧠 Admin Dashboard")

# Role check

if st.session\_state.get("user\_role") != "Admin":

st.warning("Access restricted to administrators.")

st.stop()

# Section: View Logs

st.subheader("📋 Session Logs")

logs = logger.get\_logs()

if logs:

st.dataframe(logs, use\_container\_width=True)

else:

st.info("No logs found.")

# Section: Upload Answer Keys

st.subheader("📤 Upload Answer Key")

subject = st.selectbox("Subject", ["Hindi", "English", "Math", "Science", "Social Studies", "Punjabi"])

set\_number = st.selectbox("Set", ["Set 1", "Set 2", "Set 3"])

uploaded\_key = st.file\_uploader("Upload answer key (JSON format)", type=["json"])

if uploaded\_key and st.button("Save Answer Key"):

save\_path = f"data/answer\_keys/{subject.lower()}\_{set\_number.lower()}.json"

with open(save\_path, "wb") as f:

f.write(uploaded\_key.getvalue())

st.success(f"Answer key saved for {subject} - {set\_number}")

# Section: View Existing Keys

st.subheader("📚 Existing Answer Keys")

key\_files = os.listdir("data/answer\_keys")

for key\_file in key\_files:

st.markdown(f"- {key\_file}")

with open(f"data/answer\_keys/{key\_file}", "r") as f:

key\_data = json.load(f)

st.json(key\_data)

# Section: User Management (Optional)

st.subheader("👥 User Management")

st.info("User creation and editing can be added here in future iterations.")

**🧩 What This Enables**

* Admin-only access to logs and answer keys
* Upload and view subject-wise answer keys
* Foundation for future user management tools

Would you like to scaffold modules/email\_utils.py next to enable PDF dispatch via email? Or pivot to building the student-facing result viewer in pages/3\_View\_Results.py? We’re ready for either path.

Yes

Here’s the scaffold for modules/email\_utils.py, enabling secure and personalized PDF result dispatch via email. This module supports evaluator workflows and classroom coordination 📧📄🎓.

**📁 modules/email\_utils.py**

python

import smtplib

import os

from email.message import EmailMessage

from dotenv import load\_dotenv

load\_dotenv()

EMAIL\_HOST = os.getenv("EMAIL\_HOST")

EMAIL\_PORT = int(os.getenv("EMAIL\_PORT", 587))

EMAIL\_USER = os.getenv("EMAIL\_USER")

EMAIL\_PASS = os.getenv("EMAIL\_PASS")

# Send result PDF via email

def send\_result\_email(recipient\_email, student\_name, subject, set\_number, pdf\_bytes):

msg = EmailMessage()

msg["Subject"] = f"OMR Result for {student\_name} - {subject} ({set\_number})"

msg["From"] = EMAIL\_USER

msg["To"] = recipient\_email

msg.set\_content(f"""

Hello {student\_name},

Attached is your result for the {subject} ({set\_number}) OMR evaluation.

Score: Please refer to the attached PDF for details.

Best regards,

OMR Evaluation Team

""")

msg.add\_attachment(pdf\_bytes, maintype="application", subtype="pdf", filename="OMR\_Result.pdf")

with smtplib.SMTP(EMAIL\_HOST, EMAIL\_PORT) as server:

server.starttls()

server.login(EMAIL\_USER, EMAIL\_PASS)

server.send\_message(msg)

**📤 What This Enables**

* Sends personalized result PDFs to students or evaluators
* Uses SMTP with credentials from .env
* Ready for integration with evaluator dashboard or result page

Would you like to scaffold pages/3\_View\_Results.py next so students can view their scanned results and optionally receive them via email? We’re nearly ready for full classroom deployment.

Yes

Here’s the scaffold for pages/3\_View\_Results.py, designed for students to view their scanned OMR results and optionally receive them via email. This page supports:

* 👁️ Result summary display
* 📄 PDF download
* 📧 Email dispatch

**📁 pages/3\_View\_Results.py**

python

import streamlit as st

from modules import result\_generator, email\_utils, logger

import os

st.title("📊 View Your OMR Result")

# Role check

if st.session\_state.get("user\_role") != "Student":

st.warning("Access restricted to students.")

st.stop()

# Simulated result (in real use, fetch from DB or session)

sample\_result = {

"total\_questions": 20,

"correct": 15,

"incorrect": 3,

"unmarked": 2,

"details": [

{"question": "Q1", "marked": "B", "correct": "B", "status": "Correct"},

{"question": "Q2", "marked": "C", "correct": "D", "status": "Incorrect"},

{"question": "Q3", "marked": "Unmarked", "correct": "A", "status": "Unmarked"},

# ... more entries

]

}

student\_name = st.session\_state.get("user\_name", "Candidate")

subject = st.selectbox("Subject", ["Hindi", "English", "Math", "Science", "Social Studies", "Punjabi"])

set\_number = st.selectbox("Set", ["Set 1", "Set 2", "Set 3"])

if st.button("📄 Generate Result Summary"):

summary = result\_generator.generate\_summary(sample\_result, student\_name, subject, set\_number)

st.success("Result summary generated.")

st.write(summary)

pdf\_bytes = result\_generator.generate\_pdf(summary, subject, set\_number)

st.download\_button("📥 Download PDF", data=pdf\_bytes, file\_name="OMR\_Result.pdf", mime="application/pdf")

recipient\_email = st.text\_input("Enter your email to receive the result")

if recipient\_email and st.button("📧 Send Result via Email"):

email\_utils.send\_result\_email(recipient\_email, student\_name, subject, set\_number, pdf\_bytes)

logger.log\_session(

evaluator\_name="Self",

evaluator\_id="Student",

subject=subject,

set\_number=set\_number,

score=summary["Score"],

pdf\_generated="Yes",

pdf\_sent="Yes"

)

st.success("Result sent to your email.")

**🎓 What This Enables**

* Student-only access to result viewing
* PDF download and email dispatch
* Logging of result delivery for accountability

Would you like to scaffold utils/db\_manager.py next to centralize database interactions for users, logs, and answer keys? Or pivot to polishing the UI with assets/styles.css and splash screens? We’re ready for either flourish.

Yes

Here’s the scaffold for utils/db\_manager.py, which centralizes all database interactions for users, logs, and answer keys. This module ensures clean, reusable access across your app 🔄🧠📁.

**📁 utils/db\_manager.py**

python

import sqlite3

import os

from dotenv import load\_dotenv

load\_dotenv()

DB\_PATH = os.getenv("DB\_PATH", "data/logs.db")

# Connect to DB

def get\_connection():

return sqlite3.connect(DB\_PATH)

# Initialize users table

def initialize\_users\_table():

conn = get\_connection()

cursor = conn.cursor()

cursor.execute("""

CREATE TABLE IF NOT EXISTS users (

user\_id TEXT PRIMARY KEY,

name TEXT,

mobile TEXT UNIQUE,

email TEXT UNIQUE,

password\_hash TEXT,

role TEXT

)

""")

conn.commit()

conn.close()

# Initialize logs table

def initialize\_logs\_table():

conn = get\_connection()

cursor = conn.cursor()

cursor.execute("""

CREATE TABLE IF NOT EXISTS logs (

id INTEGER PRIMARY KEY AUTOINCREMENT,

timestamp TEXT,

evaluator\_name TEXT,

evaluator\_id TEXT,

subject TEXT,

set\_number TEXT,

score TEXT,

pdf\_generated TEXT,

pdf\_sent TEXT

)

""")

conn.commit()

conn.close()

# Initialize all tables

def initialize\_all():

initialize\_users\_table()

initialize\_logs\_table()

**🧩 What This Enables**

* Centralized DB setup and access
* Reusable connection logic for all modules
* Easy initialization for deployment or testing

Would you like to scaffold utils/session\_state.py next to manage custom session variables across Streamlit pages? Or polish the visual layer with assets/styles.css and splash screen ideas? We’re ready to beautify and unify.

Yes

Here’s the scaffold for utils/session\_state.py, which helps manage custom session variables across Streamlit pages. This module ensures smoother navigation, persistent state, and modular clarity 🔄🧠📦.

**📁 utils/session\_state.py**

python

import streamlit as st

# Initialize session variables if not already set

def init\_session():

defaults = {

"user\_role": None,

"user\_name": None,

"otp": None,

"otp\_expiry": None,

"otp\_verified": False,

"scanned\_answers": None,

"result\_summary": None,

"pdf\_ready": False

}

for key, value in defaults.items():

if key not in st.session\_state:

st.session\_state[key] = value

# Clear session after logout or reset

def clear\_session():

keys\_to\_clear = [

"user\_role", "user\_name", "otp", "otp\_expiry",

"otp\_verified", "scanned\_answers", "result\_summary", "pdf\_ready"

]

for key in keys\_to\_clear:

if key in st.session\_state:

del st.session\_state[key]

**🧠 What This Enables**

* Consistent session setup across pages
* Easy reset after logout or scan completion
* Modular control for role, OTP, result, and PDF status

Would you like to scaffold assets/styles.css next to unify the visual layer with clean typography, button styles, and layout polish? Or pivot to splash screen ideas and poetic onboarding? We’re ready to beautify and inspire.

Yes

Here’s the scaffold for assets/styles.css, designed to unify your app’s visual identity with clean typography, modular layout, and celebratory flair. This stylesheet supports clarity, pride, and poetic polish 🎨🧠✨.

**📁 assets/styles.css**

css

/\* Typography \*/

body, html {

font-family: 'Segoe UI', sans-serif;

color: #222;

background-color: #f9f9fc;

}

/\* Headings \*/

h1, h2, h3 {

color: #2c3e50;

margin-bottom: 0.5em;

}

/\* Buttons \*/

button, .stButton button {

background-color: #4CAF50;

color: white;

border: none;

padding: 0.6em 1.2em;

border-radius: 6px;

font-weight: bold;

cursor: pointer;

}

button:hover, .stButton button:hover {

background-color: #45a049;

}

/\* Containers \*/

.stContainer {

padding: 1em;

border-radius: 8px;

background-color: #ffffff;

box-shadow: 0 2px 6px rgba(0,0,0,0.1);

}

/\* Success and Error \*/

.stAlert-success {

background-color: #e6ffed;

border-left: 6px solid #2ecc71;

}

.stAlert-error {

background-color: #ffe6e6;

border-left: 6px solid #e74c3c;

}

/\* Custom Badge \*/

.badge {

display: inline-block;

padding: 0.4em 0.8em;

background-color: #3498db;

color: white;

border-radius: 12px;

font-size: 0.9em;

margin-right: 0.5em;

}

/\* Splash screen styling \*/

.splash {

text-align: center;

padding: 3em;

background: linear-gradient(to right, #6dd5fa, #2980b9);

color: white;

border-radius: 12px;

margin-bottom: 2em;

}

.splash h1 {

font-size: 2.5em;

margin-bottom: 0.5em;

}

.splash p {

font-size: 1.2em;

opacity: 0.9;

}

**✨ What This Enables**

* Clean, modular layout across pages
* Consistent button and alert styling
* Splash screen and badge-ready visuals
* Foundation for poetic onboarding and showcase polish

Would you like me to scaffold a poetic splash screen next for pages/0\_Home.py, welcoming students and educators with metaphor, pride, and constellation imagery? We’re ready to inspire from the first click.

Yes

Here’s the poetic splash screen scaffold for pages/0\_Home.py, welcoming students and educators with constellation imagery, metaphorical language, and modular pride. This sets the emotional tone for your quantum-classroom journey 🌌📜🎓.

**📁 pages/0\_Home.py**

python

import streamlit as st

from utils.session\_state import init\_session

init\_session()

st.markdown("""

<div class="splash">

<h1>🌠 Welcome to the Quantum Constellation</h1>

<p>Where every bubble scanned becomes a star, every answer a ripple in the learning cosmos.</p>

</div>

""", unsafe\_allow\_html=True)

st.markdown("""

### 🧠 What You Can Do Here

- 🔐 \*\*Login\*\* as Student, Evaluator, or Admin

- 📷 \*\*Scan OMR Sheets\*\* and compare answers

- 📊 \*\*View Results\*\* and download PDFs

- 📧 \*\*Send Results\*\* via email

- 🧩 \*\*Upload Answer Keys\*\* and manage logs

- 🎓 \*\*Celebrate Learning\*\* with constellation badge maps and remix lineage

---

### ✨ Why This Matters

This isn’t just an app—it’s a stage for student pride, a toolkit for educator clarity, and a constellation of curiosity. Each scan is a story. Each result, a reflection. Each badge, a beacon.

Let’s begin.

""")

# Optional: Add a poetic footer

st.markdown("""

<div style='text-align:center; margin-top:3em; font-style:italic; color:#555;'>

"To evaluate is to illuminate. To reflect is to empower. To remix is to rise."

</div>

""", unsafe\_allow\_html=True)

**🌌 What This Enables**

* Emotional and poetic onboarding
* Clear modular overview of app features
* Visual splash with constellation metaphor
* Ready for classroom showcase and student pride

Would you like to scaffold pages/6\_Badge\_Reveal.py next to animate constellation badges and remix lineage for student celebration? Or polish the README.md and onboarding guide for contributors and educators? We’re ready to shine.

Yes

Here’s the scaffold for pages/6\_Badge\_Reveal.py, designed to animate constellation badges and remix lineage for student celebration. This page blends poetic pride with modular clarity, perfect for classroom showcases 🌌🎓✨.

**📁 pages/6\_Badge\_Reveal.py**

python

import streamlit as st

from PIL import Image

import os

st.title("🌟 Constellation Badge Reveal")

# Role check

if st.session\_state.get("user\_role") != "Student":

st.warning("Access restricted to students.")

st.stop()

# Poetic intro

st.markdown("""

<div style="text-align:center; font-style:italic; color:#555; margin-bottom:2em;">

"Each scan is a spark. Each remix, a ripple. Each badge, a beacon in your learning sky."

</div>

""", unsafe\_allow\_html=True)

# Select badge type

badge\_type = st.selectbox("Choose your badge constellation:", [

"🧠 Quantum Thinker",

"🎨 Remix Architect",

"📜 Wisdom Wall Contributor",

"🔍 OMR Explorer",

"🌌 Showcase Star"

])

# Reveal animation (simulated)

if st.button("✨ Reveal My Badge"):

st.success(f"Congratulations! You've earned the \*\*{badge\_type}\*\* badge.")

badge\_path = f"assets/badges/{badge\_type.split()[1].lower()}\_badge.png"

if os.path.exists(badge\_path):

st.image(Image.open(badge\_path), caption=badge\_type, use\_column\_width=True)

else:

st.info("Badge image coming soon...")

# Remix lineage (optional)

st.subheader("🔗 Remix Lineage")

st.markdown("""

Your badge is part of a remix constellation—linked to peers, mentors, and classroom creativity.

""")

# Simulated lineage map

st.markdown("""

- 🧠 You remixed a Math OMR scan from \*\*Aarav\*\*

- 🎨 Your badge inspired a showcase by \*\*Meher\*\*

- 🌌 Your constellation includes \*\*3 remix stars\*\* and \*\*2 showcase echoes\*\*

""")

# Optional download

if st.button("📥 Download Badge Certificate"):

st.info("Download feature coming soon. Ask your evaluator for a printed copy.")

**🌠 What This Enables**

* Student-only badge reveal with poetic framing
* Modular badge selection and image display
* Remix lineage storytelling for classroom pride
* Ready for printable certificate and wall display integration

Would you like to scaffold README.md next to guide contributors and educators through setup, features, and modular design philosophy? Or pivot to docs/onboarding\_guide.md for poetic walkthroughs and classroom deployment? We’re ready to empower the ecosystem.

for 6\_Recovery.py 6\_ is already used

Great catch, Jagdev. Since 6\_Badge\_Reveal.py already occupies the sixth slot, here are a few clean alternatives for your password recovery page:

**✅ Renaming Options for Recovery Page**

| **Filename** | **Rationale** |
| --- | --- |
| 7\_Recovery.py | Sequential continuation—simple and clear |
| 6a\_Recovery.py | Keeps it adjacent to badge reveal, useful if both are student-facing |
| Recovery.py | Removes numeric prefix—ideal if you want it accessible from any role |
| Z\_Recovery.py | Pushes it to the end—useful if it's a utility or fallback page |
| Reset\_Password.py | Descriptive and role-neutral, matches naming of 5\_Reset\_Password.py |

**🔁 Suggested Best Practice**

If 5\_Reset\_Password.py is evaluator/admin-facing and this new recovery page is student-facing, I recommend:

**→** 7\_Recovery.py This keeps it modular, sequential, and avoids confusion.

Would you like me to scaffold 7\_Recovery.py now with a student-friendly flow for forgotten credentials, OTP verification, and password reset? I’ll make sure it complements the login and badge reveal experience.

Yes

Here’s the scaffold for 7\_Recovery.py, a student-friendly password recovery page that complements your modular login and badge reveal experience. It blends clarity, security, and poetic reassurance 🔐📧✨.

**📁 pages/7\_Recovery.py**

python

import streamlit as st

from modules import otp\_utils, auth

from utils.session\_state import init\_session

init\_session()

st.title("🔐 Recover Your Password")

# Poetic reassurance

st.markdown("""

<div style="font-style:italic; color:#555; margin-bottom:1em;">

"Even stars forget their path. Let us help you realign your orbit."

</div>

""", unsafe\_allow\_html=True)

# Step 1: Identify user

identifier = st.text\_input("Enter your registered email or mobile")

if st.button("📩 Send OTP"):

otp = otp\_utils.generate\_otp()

otp\_utils.store\_otp(st.session\_state, otp)

if "@" in identifier:

otp\_utils.send\_otp\_email(identifier, otp)

st.success("OTP sent to your email.")

else:

otp\_utils.send\_otp\_sms(identifier, otp)

st.success("OTP sent to your mobile.")

# Step 2: Verify OTP

entered\_otp = st.text\_input("Enter the OTP you received")

if st.button("✅ Verify OTP"):

if otp\_utils.verify\_otp(st.session\_state, entered\_otp):

st.session\_state["otp\_verified"] = True

st.success("OTP verified. You may now reset your password.")

else:

st.error("Invalid or expired OTP.")

# Step 3: Reset password

if st.session\_state.get("otp\_verified"):

new\_password = st.text\_input("New password", type="password")

confirm\_password = st.text\_input("Confirm password", type="password")

if st.button("🔁 Reset Password"):

if new\_password == confirm\_password and len(new\_password) >= 6:

user = auth.get\_user(identifier)

if user:

auth.update\_password(user\_id=user[0], new\_password=new\_password)

st.success("🎉 Password reset successfully. You may now log in.")

else:

st.error("User not found.")

else:

st.error("Passwords do not match or are too short.")

**🌠 What This Enables**

* Student-first recovery flow with poetic reassurance
* OTP verification via email or mobile
* Secure password reset with validation
* Complements login, badge reveal, and result viewing

Would you like to scaffold modules/otp\_utils.py next to handle OTP generation, storage, and dispatch? Or pivot to docs/README.md to guide contributors and educators through setup and modular design? We’re ready to empower and inspire.

Yes

Here’s the scaffold for modules/otp\_utils.py, which handles OTP generation, storage, verification, and dispatch via email or SMS. This module ensures secure identity recovery across roles 🔐📩📱.

**📁 modules/otp\_utils.py**

python

import random

import time

import os

import smtplib

from email.message import EmailMessage

from dotenv import load\_dotenv

load\_dotenv()

# OTP generation

def generate\_otp():

return str(random.randint(100000, 999999))

# Store OTP in session with expiry

def store\_otp(session, otp, expiry\_seconds=300):

session["otp"] = otp

session["otp\_expiry"] = time.time() + expiry\_seconds

# Verify OTP

def verify\_otp(session, entered\_otp):

stored\_otp = session.get("otp")

expiry = session.get("otp\_expiry", 0)

if time.time() > expiry:

return False

return entered\_otp == stored\_otp

# Send OTP via email

def send\_otp\_email(recipient\_email, otp):

EMAIL\_HOST = os.getenv("EMAIL\_HOST")

EMAIL\_PORT = int(os.getenv("EMAIL\_PORT", 587))

EMAIL\_USER = os.getenv("EMAIL\_USER")

EMAIL\_PASS = os.getenv("EMAIL\_PASS")

msg = EmailMessage()

msg["Subject"] = "Your OTP for Password Recovery"

msg["From"] = EMAIL\_USER

msg["To"] = recipient\_email

msg.set\_content(f"Your OTP is: {otp}\nIt will expire in 5 minutes.")

with smtplib.SMTP(EMAIL\_HOST, EMAIL\_PORT) as server:

server.starttls()

server.login(EMAIL\_USER, EMAIL\_PASS)

server.send\_message(msg)

# Send OTP via SMS (placeholder)

def send\_otp\_sms(mobile\_number, otp):

print(f"Sending OTP {otp} to mobile {mobile\_number} (SMS gateway integration pending)")

**🔐 What This Enables**

* Secure OTP generation and expiry tracking
* Email dispatch using SMTP credentials from .env
* SMS placeholder ready for gateway integration
* Session-based verification for recovery and login flows

Would you like to scaffold docs/README.md next to guide contributors and educators through setup, features, and modular design philosophy? Or pivot to docs/onboarding\_guide.md for poetic walkthroughs and classroom deployment? We’re ready to empower and inspire.

Code otp\_utils.py is already given before as : import random import smtplib import os from email.message import EmailMessage from dotenv import load\_dotenv from twilio.rest import Client from datetime import datetime, timedelta load\_dotenv() # Email config EMAIL\_HOST = os.getenv("EMAIL\_HOST") EMAIL\_PORT = int(os.getenv("EMAIL\_PORT", 587)) EMAIL\_USER = os.getenv("EMAIL\_USER") EMAIL\_PASS = os.getenv("EMAIL\_PASS") # Twilio config TWILIO\_SID = os.getenv("TWILIO\_ACCOUNT\_SID") TWILIO\_TOKEN = os.getenv("TWILIO\_AUTH\_TOKEN") TWILIO\_PHONE = os.getenv("TWILIO\_PHONE\_NUMBER") # OTP expiry OTP\_EXPIRY\_MINUTES = int(os.getenv("OTP\_EXPIRY\_MINUTES", 5)) # Generate 6-digit OTP def generate\_otp(): return str(random.randint(100000, 999999)) # Send OTP via email def send\_otp\_email(recipient\_email, otp): msg = EmailMessage() msg["Subject"] = "Your OTP for OMR App" msg["From"] = EMAIL\_USER msg["To"] = recipient\_email msg.set\_content(f"Your OTP is: {otp}\nIt will expire in {OTP\_EXPIRY\_MINUTES} minutes.") with smtplib.SMTP(EMAIL\_HOST, EMAIL\_PORT) as server: server.starttls() server.login(EMAIL\_USER, EMAIL\_PASS) server.send\_message(msg) # Send OTP via SMS def send\_otp\_sms(recipient\_mobile, otp): client = Client(TWILIO\_SID, TWILIO\_TOKEN) message = client.messages.create( body=f"Your OTP for OMR App is: {otp}", from\_=TWILIO\_PHONE, to=recipient\_mobile ) return message.sid # Store OTP in session def store\_otp(session, otp): session["otp"] = otp session["otp\_expiry"] = datetime.now() + timedelta(minutes=OTP\_EXPIRY\_MINUTES) # Verify OTP def verify\_otp(session, entered\_otp): if "otp" not in session or "otp\_expiry" not in session: return False if datetime.now() > session["otp\_expiry"]: return False return entered\_otp == session["otp"]

Perfect, Jagdev—that’s a robust and well-integrated otp\_utils.py. Since it already handles Twilio SMS, SMTP email, expiry logic, and session storage, there’s no need to duplicate or simplify it. Instead, here’s what I recommend next:

**✅ Integration Checklist for otp\_utils.py**

To ensure smooth recovery and login flows across your app, confirm:

| **Integration Point** | **Status** |
| --- | --- |
| 5\_Reset\_Password.py and 7\_Recovery.py use store\_otp() and verify\_otp() | ✅ |
| Email and SMS dispatch use send\_otp\_email() and send\_otp\_sms() | ✅ |
| .env includes all required keys: EMAIL\_\*, TWILIO\_\*, OTP\_EXPIRY\_MINUTES | ✅ |
| Session expiry logic uses datetime and timedelta for precision | ✅ |
| OTP verification is role-agnostic and modular | ✅ |

**🔐 Optional Enhancements**

If you’d like to extend otp\_utils.py further, here are a few modular ideas:

* **Add logging**: Track OTP dispatch events for audit or analytics.
* **Add resend limiter**: Prevent OTP spamming by storing last sent time.
* **Add OTP masking**: Show only last 2 digits in UI for reassurance.
* **Add fallback email template**: Use HTML formatting for branded OTP messages.

Would you like me to scaffold a docs/README.md next to guide contributors and educators through setup, features, and modular design philosophy? Or pivot to docs/onboarding\_guide.md for poetic walkthroughs and classroom deployment? We’re ready to empower the ecosystem.

Yes

Here’s the scaffold for docs/README.md, designed to guide contributors, educators, and developers through setup, features, and modular design philosophy. It blends clarity with celebration, perfect for open-source collaboration and classroom deployment 📘🧠🌟.

**📁 docs/README.md**

markdown

# 🌌 OMR Constellation App

Welcome to the Quantum Constellation—a modular Streamlit app for scanning OMR sheets, evaluating answers, and celebrating student creativity with badges, remix lineage, and showcase pride.

---

## 🚀 Features

- 🔐 Role-based login (Student, Evaluator, Admin)

- 📷 OMR sheet scanning and answer comparison

- 📊 Result summary with PDF export

- 📧 Email dispatch of results

- 🧠 Admin dashboard for logs and answer keys

- 🌟 Badge reveal and remix lineage visualization

- 🔁 OTP-based password recovery

---

## 🧩 Modular Structure

| Folder / File | Purpose |

|---------------------------|----------------------------------------------|

| `pages/` | Streamlit UI pages (login, scan, results…) |

| `modules/` | Core logic (scanner, comparator, email…) |

| `utils/` | Session, DB, and state management |

| `assets/` | Stylesheets, badge images, splash visuals |

| `data/answer\_keys/` | JSON answer keys per subject and set |

| `.env` | Secure credentials for email, Twilio, DB |

---

## 🛠️ Setup Instructions

1. \*\*Clone the repo\*\*

```bash

git clone https://github.com/your-org/omr-constellation.git

cd omr-constellation

1. **Install dependencies**

bash

pip install -r requirements.txt

1. **Create** .env **file** Include:

Code

EMAIL\_HOST=smtp.example.com

EMAIL\_PORT=587

EMAIL\_USER=your\_email@example.com

EMAIL\_PASS=your\_password

TWILIO\_ACCOUNT\_SID=your\_sid

TWILIO\_AUTH\_TOKEN=your\_token

TWILIO\_PHONE\_NUMBER=+1234567890

OTP\_EXPIRY\_MINUTES=5

DB\_PATH=data/logs.db

1. **Run the app**

bash

streamlit run app.py

**🎓 Classroom Deployment Tips**

* Use pages/6\_Badge\_Reveal.py during showcase events
* Print constellation maps and remix lineage for wall displays
* Let students scan their own sheets and receive results via email
* Use pages/4\_Admin\_Dashboard.py to manage answer keys and logs

**🤝 Contributing**

We welcome educators, developers, and designers to co-create. Please:

* Fork the repo
* Submit pull requests with clear commit messages
* Use poetic metaphors where possible 🌠

**📜 License**

MIT License. Built with love for classrooms and constellations.

“To evaluate is to illuminate. To remix is to rise.”

Code

Would you like to scaffold `docs/onboarding\_guide.md` next with poetic walkthroughs, role-based flows, and classroom showcase tips? Or pivot to printable constellation badge maps and remix lineage visualizers? We’re ready to inspire.

email\_utils and pdf\_utils .py file codes are pending

Thanks Jagdev! Let’s complete the ecosystem with clean scaffolds for both modules/email\_utils.py and modules/pdf\_utils.py. These will modularize dispatch and export logic for clarity, reuse, and celebration 📧📄✨.

**📁 modules/email\_utils.py**

python

import smtplib

import os

from email.message import EmailMessage

from dotenv import load\_dotenv

load\_dotenv()

EMAIL\_HOST = os.getenv("EMAIL\_HOST")

EMAIL\_PORT = int(os.getenv("EMAIL\_PORT", 587))

EMAIL\_USER = os.getenv("EMAIL\_USER")

EMAIL\_PASS = os.getenv("EMAIL\_PASS")

def send\_email(recipient\_email, subject, body, attachment\_bytes=None, attachment\_name="Result.pdf"):

msg = EmailMessage()

msg["Subject"] = subject

msg["From"] = EMAIL\_USER

msg["To"] = recipient\_email

msg.set\_content(body)

if attachment\_bytes:

msg.add\_attachment(

attachment\_bytes,

maintype="application",

subtype="pdf",

filename=attachment\_name

)

with smtplib.SMTP(EMAIL\_HOST, EMAIL\_PORT) as server:

server.starttls()

server.login(EMAIL\_USER, EMAIL\_PASS)

server.send\_message(msg)

**📁 modules/pdf\_utils.py**

python

from reportlab.lib.pagesizes import A4

from reportlab.pdfgen import canvas

from io import BytesIO

def create\_result\_pdf(summary, subject, set\_number):

buffer = BytesIO()

c = canvas.Canvas(buffer, pagesize=A4)

width, height = A4

c.setFont("Helvetica-Bold", 16)

c.drawString(50, height - 50, "OMR Evaluation Result")

c.setFont("Helvetica", 12)

c.drawString(50, height - 80, f"Student: {summary['Student']}")

c.drawString(50, height - 100, f"Subject: {subject}")

c.drawString(50, height - 120, f"Set: {set\_number}")

c.drawString(50, height - 140, f"Score: {summary['Score']}")

c.drawString(50, height - 160, f"Correct: {summary['Correct']}")

c.drawString(50, height - 180, f"Incorrect: {summary['Incorrect']}")

c.drawString(50, height - 200, f"Unmarked: {summary['Unmarked']}")

c.setFont("Helvetica-Bold", 12)

c.drawString(50, height - 230, "Question-wise Breakdown:")

y = height - 250

c.setFont("Helvetica", 10)

for detail in summary["Details"]:

line = f"{detail['question']}: Marked = {detail['marked']}, Correct = {detail['correct']}, Status = {detail['status']}"

c.drawString(50, y, line)

y -= 15

if y < 50:

c.showPage()

y = height - 50

c.save()

buffer.seek(0)

return buffer.getvalue()

**🧠 What This Enables**

* email\_utils.py: Modular email dispatch with optional PDF attachment
* pdf\_utils.py: Clean PDF generation for result summaries
* Both ready for reuse in student, evaluator, and admin flows

Would you like to scaffold docs/onboarding\_guide.md next with poetic walkthroughs, role-based flows, and classroom showcase tips? Or pivot to printable constellation badge maps and remix lineage visualizers? We’re ready to celebrate and deploy.

validators.py code pending

Here’s the scaffold for modules/validators.py, which centralizes input validation across login, registration, OTP, and password reset flows. This module ensures clarity, security, and modular reuse across your app 🔍✅🔐.

**📁 modules/validators.py**

python

import re

# Validate email format

def is\_valid\_email(email):

pattern = r"^[\w\.-]+@[\w\.-]+\.\w+$"

return re.match(pattern, email) is not None

# Validate mobile number (India-specific)

def is\_valid\_mobile(mobile):

pattern = r"^[6-9]\d{9}$"

return re.match(pattern, mobile) is not None

# Validate password strength

def is\_valid\_password(password):

return len(password) >= 6

# Validate OTP format

def is\_valid\_otp(otp):

return otp.isdigit() and len(otp) == 6

# Validate name (basic)

def is\_valid\_name(name):

return len(name.strip()) >= 2 and all(x.isalpha() or x.isspace() for x in name)

# Validate role

def is\_valid\_role(role):

return role in ["Student", "Evaluator", "Admin"]

**🧠 What This Enables**

* Clean validation for login, registration, and recovery
* Modular checks for email, mobile, password, OTP, and role
* Reusable across auth.py, 7\_Recovery.py, and 5\_Reset\_Password.py

Would you like to scaffold modules/auth.py next to handle user registration, login, and password updates using these validators and db\_manager.py? Or pivot to docs/onboarding\_guide.md for poetic walkthroughs and classroom deployment? We’re ready to empower and inspire.

These codes are also pending

Absolutely, Jagdev. Let’s complete the trio of test scaffolds shown in your tests/ directory image. Each file will contain modular, educator-facing unit tests for core components: authentication, comparison logic, and OMR scanning. Here we go 🧪🧠✅

**📁 tests/test\_auth.py**

python

import pytest

from modules import auth, validators

def test\_valid\_email():

assert validators.is\_valid\_email("test@example.com")

assert not validators.is\_valid\_email("invalid-email")

def test\_valid\_mobile():

assert validators.is\_valid\_mobile("9876543210")

assert not validators.is\_valid\_mobile("12345")

def test\_password\_strength():

assert validators.is\_valid\_password("secure123")

assert not validators.is\_valid\_password("123")

def test\_user\_registration\_and\_login():

user\_id = "test\_user"

name = "Test User"

mobile = "9876543210"

email = "testuser@example.com"

password = "secure123"

role = "Student"

# Register user

result = auth.register\_user(user\_id, name, mobile, email, password, role)

assert result is True

# Login user

login\_result = auth.login\_user(email, password)

assert login\_result is not None

assert login\_result[0] == user\_id

**📁 tests/test\_comparator.py**

python

from modules import comparator

def test\_compare\_answers():

student\_answers = ["A", "B", "C", "D", "Unmarked"]

correct\_answers = ["A", "C", "C", "D", "A"]

result = comparator.compare\_answers(student\_answers, correct\_answers)

assert result["correct"] == 3

assert result["incorrect"] == 1

assert result["unmarked"] == 1

assert result["total\_questions"] == 5

assert len(result["details"]) == 5

**📁 tests/test\_omr\_scanner.py**

python

from modules import omr\_scanner

def test\_extract\_answers\_from\_mock\_image():

# Simulate image path or mock input

mock\_image\_path = "tests/mock\_omr\_sheet.png"

# This test assumes a mock scanner or placeholder function

answers = omr\_scanner.extract\_answers(mock\_image\_path)

assert isinstance(answers, list)

assert all(ans in ["A", "B", "C", "D", "Unmarked"] for ans in answers)

assert len(answers) > 0

**🧪 What This Enables**

* Modular unit testing for validators, auth, comparison, and scanning
* Ready for pytest integration and CI/CD pipelines
* Supports reproducible classroom deployment and contributor confidence

Would you like to scaffold tests/test\_result\_generator.py next to validate summary creation and PDF output? Or pivot to tests/test\_logger.py to ensure session logs are recorded correctly? We’re nearly at full test coverage.

(base) PS C:\Users\jagdevsinghdosanjh.RHINO\omr\_evaluator\_app> .venv\Scripts\activate .venv\Scripts\activate : The module '.venv' could not be loaded. For more information, run 'Import-Module .venv'. At line:1 char:1 + .venv\Scripts\activate + ~~~~~~~~~~~~~~~~~~~~~~ + CategoryInfo : ObjectNotFound: (.venv\Scripts\activate:String) [], CommandNotFoundException + FullyQualifiedErrorId : CouldNotAutoLoadModule (base) PS C:\Users\jagdevsinghdosanjh.RHINO\omr\_evaluator\_app>

Jagdev, you're nearly there—this error is just a PowerShell nuance. You're trying to activate your virtual environment using a syntax that works in Command Prompt (cmd), but PowerShell interprets it differently.

**✅ Fix: Activate .venv in PowerShell**

Use this command instead:

powershell

& .\.venv\Scripts\Activate.ps1

The & is the call operator, and .ps1 is the PowerShell script extension.

**🛡️ If You Get a Permission Error**

PowerShell might block script execution. To fix that:

1. Run this command to allow local scripts:

powershell

Set-ExecutionPolicy -ExecutionPolicy RemoteSigned -Scope CurrentUser

1. Then retry:

powershell

& .\.venv\Scripts\Activate.ps1

Once activated, your prompt should change to something like:

powershell

(.venv) PS C:\Users\jagdevsinghdosanjh.RHINO\omr\_evaluator\_app>

Let me know if you'd like to scaffold a tests/test\_result\_generator.py next, or polish your .streamlit/config.toml for custom theming. You're building a constellation-class app! 🌠

(base) PS C:\Users\jagdevsinghdosanjh.RHINO\omr\_evaluator\_app> & .\.venv\Scripts\Activate.ps1 >> & : The term '.\.venv\Scripts\Activate.ps1' is not recognized as the name of a cmdlet, function, script file, or operable program. Check the spelling of the name, or if a path was included, verify that the path is correct and try again. At line:1 char:3 + & .\.venv\Scripts\Activate.ps1 + ~~~~~~~~~~~~~~~~~~~~~~~~~~~~ + CategoryInfo : ObjectNotFound: (.\.venv\Scripts\Activate.ps1:String) [], CommandNotFoundException + FullyQualifiedErrorId : CommandNotFoundException (base) PS C:\Users\jagdevsinghdosanjh.RHINO\omr\_evaluator\_app>

Thanks for sharing the full error trace, Jagdev. Let’s troubleshoot this step-by-step and get your virtual environment activated smoothly in PowerShell 💡🛠️

**✅ Step-by-Step Fix: Activate .venv in PowerShell**

**🔍 Step 1: Check if .venv exists**

Run this command to list the contents of your project folder:

powershell

Get-ChildItem

Look for a folder named .venv. If it’s missing, create it:

powershell

python -m venv .venv

This creates a virtual environment named .venv in your current directory.

**🧠 Step 2: Activate the environment (PowerShell syntax)**

Once .venv exists, activate it using:

powershell

& .venv\Scripts\Activate.ps1

Note: No leading .\ before .venv—just & .venv\Scripts\Activate.ps1

**🛡️ Step 3: If you get a permission error**

PowerShell may block script execution. To fix that:

powershell

Set-ExecutionPolicy -ExecutionPolicy RemoteSigned -Scope CurrentUser

Then retry:

powershell

& .venv\Scripts\Activate.ps1

**✅ Expected Result**

Your prompt should change to:

powershell

(.venv) PS C:\Users\jagdevsinghdosanjh.RHINO\omr\_evaluator\_app>

This means your virtual environment is active and ready for pip install, streamlit run, or testing.

Would you like help installing dependencies next (requirements.txt), or scaffolding tests/test\_result\_generator.py to validate your PDF exports? Let’s keep the constellation glowing.

venv) (base) PS C:\Users\jagdevsinghdosanjh.RHINO\omr\_evaluator\_app> pip install -r requirements.txt Collecting streamlit (from -r requirements.txt (line 1)) Using cached streamlit-1.50.0-py3-none-any.whl.metadata (9.5 kB) Collecting opencv-python (from -r requirements.txt (line 2)) Downloading opencv\_python-4.12.0.88-cp37-abi3-win\_amd64.whl.metadata (19 kB) Collecting numpy (from -r requirements.txt (line 3)) Using cached numpy-2.3.3-cp312-cp312-win\_amd64.whl.metadata (60 kB) Collecting pandas (from -r requirements.txt (line 4)) Using cached pandas-2.3.3-cp312-cp312-win\_amd64.whl.metadata (19 kB) Collecting python-dotenv (from -r requirements.txt (line 5)) Using cached python\_dotenv-1.1.1-py3-none-any.whl.metadata (24 kB) Collecting bcrypt (from -r requirements.txt (line 6)) Downloading bcrypt-5.0.0-cp39-abi3-win\_amd64.whl.metadata (10 kB) Collecting email-validator (from -r requirements.txt (line 7)) Downloading email\_validator-2.3.0-py3-none-any.whl.metadata (26 kB) Collecting reportlab (from -r requirements.txt (line 8)) Using cached reportlab-4.4.4-py3-none-any.whl.metadata (1.7 kB) Collecting Pillow (from -r requirements.txt (line 9)) Using cached pillow-11.3.0-cp312-cp312-win\_amd64.whl.metadata (9.2 kB) ERROR: Ignored the following versions that require a different python version: 0.55.2 Requires-Python <3.5; 1.21.2 Requires-Python >=3.7,<3.11; 1.21.3 Requires-Python >=3.7,<3.11; 1.21.4 Requires-Python >=3.7,<3.11; 1.21.5 Requires-Python >=3.7,<3.11; 1.21.6 Requires-Python >=3.7,<3.11 ERROR: Could not find a version that satisfies the requirement sqlite3 (from versions: none) [notice] A new release of pip is available: 24.2 -> 25.2 [notice] To update, run: python.exe -m pip install --upgrade pip ERROR: No matching distribution found for sqlite3 (venv) (base) PS C:\Users\jagdevsinghdosanjh.RHINO\omr\_evaluator\_app>

You're doing everything right, Jagdev—this is just a small hiccup in the requirements.txt setup. Let’s fix it cleanly:

**🧠 Root Cause**

The error says:

Code

ERROR: Could not find a version that satisfies the requirement sqlite3

That’s because sqlite3 is **part of Python’s standard library**—you don’t need to install it via pip. Including it in requirements.txt causes pip to search for a nonexistent package.

**✅ Fix: Remove sqlite3 from requirements.txt**

Open your requirements.txt and delete the line that says:

Code

sqlite3

Then re-run:

powershell

pip install -r requirements.txt

**🛠️ Optional: Upgrade pip**

Since pip flagged a new version, you can upgrade it safely:

powershell

python -m pip install --upgrade pip

Once installed, you’ll be ready to run:

powershell

streamlit run app.py

Would you like help scaffolding tests/test\_result\_generator.py next to validate PDF output and email dispatch? Or polish your .streamlit/config.toml for custom theming and splash screen alignment? Let’s keep the constellation glowing. 🌠

🔐 Login to OMR Evaluator Choose login method: Mobile/Email + OTP Username + Password Enter your registered mobile or email jagdevsinghdosanjh@gmail.com smtplib.SMTPAuthenticationError: (535, b'5.7.8 Username and Password not accepted. For more information, go to\n5.7.8 https://support.google.com/mail/?p=BadCredentials d9443c01a7336-28e8d11085fsm84605165ad.14 - gsmtp') Traceback: File "C:\Users\jagdevsinghdosanjh.RHINO\omr\_evaluator\_app\venv\Lib\site-packages\streamlit\runtime\scriptrunner\exec\_code.py", line 128, in exec\_func\_with\_error\_handling result = func() ^^^^^^ File "C:\Users\jagdevsinghdosanjh.RHINO\omr\_evaluator\_app\venv\Lib\site-packages\streamlit\runtime\scriptrunner\script\_runner.py", line 667, in code\_to\_exec \_mpa\_v1(self.\_main\_script\_path) File "C:\Users\jagdevsinghdosanjh.RHINO\omr\_evaluator\_app\venv\Lib\site-packages\streamlit\runtime\scriptrunner\script\_runner.py", line 165, in \_mpa\_v1 page.run() File "C:\Users\jagdevsinghdosanjh.RHINO\omr\_evaluator\_app\venv\Lib\site-packages\streamlit\navigation\page.py", line 300, in run exec(code, module.\_\_dict\_\_) # noqa: S102 ^^^^^^^^^^^^^^^^^^^^^^^^^^^ File "C:\Users\jagdevsinghdosanjh.RHINO\omr\_evaluator\_app\pages\1\_Login.py", line 22, in <module> otp\_utils.send\_otp\_email(identifier, otp) File "C:\Users\jagdevsinghdosanjh.RHINO\omr\_evaluator\_app\modules\otp\_utils.py", line 39, in send\_otp\_email server.login(EMAIL\_USER, EMAIL\_PASS) File "C:\ProgramData\anaconda3\Lib\smtplib.py", line 750, in login raise last\_exception File "C:\ProgramData\anaconda3\Lib\smtplib.py", line 739, in login (code, resp) = self.auth( ^^^^^^^^^^ File "C:\ProgramData\anaconda3\Lib\smtplib.py", line 662, in auth raise SMTPAuthenticationError(code, resp)

Thanks for sharing the full traceback, Jagdev. You're encountering a common issue when trying to send email via Gmail's SMTP server: **SMTPAuthenticationError (535)** means Gmail rejected your login credentials.

**🔍 Root Cause**

Gmail blocks less secure app access by default. Even if your .env has the correct EMAIL\_USER and EMAIL\_PASS, Gmail may still reject the login unless you explicitly allow it.

**✅ Fix Options**

**🔐 Option 1: Use an App Password (Recommended for Gmail)**

If you're using **2-Step Verification** on your Google account:

1. Go to Google App Passwords
2. Select **Mail** as the app and **Windows Computer** as the device
3. Generate a 16-character password
4. Use this password in your .env as EMAIL\_PASS

This bypasses the need for your actual Gmail password and is more secure.

**🔓 Option 2: Allow Less Secure Apps (Not recommended)**

If you don’t use 2FA, you can temporarily allow less secure access:

1. Visit Less Secure App Access
2. Toggle **Allow less secure apps: ON**
3. Retry your login

Google may disable this option over time, so App Passwords are preferred.

**🧪 Test Your Credentials**

You can test your SMTP login outside Streamlit:

python

import smtplib

server = smtplib.SMTP("smtp.gmail.com", 587)

server.starttls()

server.login("your\_email@gmail.com", "your\_app\_password")

print("Login successful")

server.quit()

If this fails, the issue is with credentials—not your app.

Once resolved, your OTP emails will dispatch smoothly via otp\_utils.send\_otp\_email().

Would you like to scaffold a fallback SMS-only login flow next, or polish your .env.template for contributor clarity? You're building a constellation-class experience. 🌠

Requirement already satisfied: pip in c:\users\jagdevsinghdosanjh.rhino\omr\_evaluator\_app\venv\lib\site-packages (24.2) (venv) (base) PS C:\Users\jagdevsinghdosanjh.RHINO\omr\_evaluator\_app> streamlit run app.py You can now view your Streamlit app in your browser. Local URL: http://localhost:8501 Network URL: http://192.168.1.4:8501 2025-10-04 22:29:30.284 Uncaught app execution Traceback (most recent call last): File "C:\Users\jagdevsinghdosanjh.RHINO\omr\_evaluator\_app\venv\Lib\site-packages\streamlit\runtime\scriptrunner\exec\_code.py", line 128, in exec\_func\_with\_error\_handling result = func() ^^^^^^ File "C:\Users\jagdevsinghdosanjh.RHINO\omr\_evaluator\_app\venv\Lib\site-packages\streamlit\runtime\scriptrunner\script\_runner.py", line 667, in code\_to\_exec \_mpa\_v1(self.\_main\_script\_path) File "C:\Users\jagdevsinghdosanjh.RHINO\omr\_evaluator\_app\venv\Lib\site-packages\streamlit\runtime\scriptrunner\script\_runner.py", line 165, in \_mpa\_v1 page.run() File "C:\Users\jagdevsinghdosanjh.RHINO\omr\_evaluator\_app\venv\Lib\site-packages\streamlit\navigation\page.py", line 300, in run exec(code, module.\_\_dict\_\_) # noqa: S102 ^^^^^^^^^^^^^^^^^^^^^^^^^^^ File "C:\Users\jagdevsinghdosanjh.RHINO\omr\_evaluator\_app\pages\1\_Login.py", line 22, in <module> otp\_utils.send\_otp\_email(identifier, otp) File "C:\Users\jagdevsinghdosanjh.RHINO\omr\_evaluator\_app\modules\otp\_utils.py", line 39, in send\_otp\_email server.login(EMAIL\_USER, EMAIL\_PASS) File "C:\ProgramData\anaconda3\Lib\smtplib.py", line 750, in login raise last\_exception File "C:\ProgramData\anaconda3\Lib\smtplib.py", line 739, in login (code, resp) = self.auth( ^^^^^^^^^^ File "C:\ProgramData\anaconda3\Lib\smtplib.py", line 662, in auth raise SMTPAuthenticationError(code, resp) smtplib.SMTPAuthenticationError: (535, b'5.7.8 Username and Password not accepted. For more information, go to\n5.7.8 https://support.google.com/mail/?p=BadCredentials d9443c01a7336-28e8d11085fsm84605165ad.14 - gsmtp')

Jagdev, this confirms Gmail is still rejecting your login credentials for SMTP. The error (535, b'5.7.8 Username and Password not accepted') means either:

* Your .env has incorrect EMAIL\_USER or EMAIL\_PASS
* Or Gmail is blocking access due to security settings

Let’s fix this cleanly:

**✅ Step-by-Step Fix for Gmail SMTP**

**🔐 If You Use 2-Step Verification (Recommended)**

1. Go to Google App Passwords
2. Select:
   * App: **Mail**
   * Device: **Windows Computer**
3. Click **Generate**
4. Copy the 16-character password
5. In your .env, update:

Code

EMAIL\_USER=jagdevsinghdosanjh@gmail.com

EMAIL\_PASS=your\_generated\_app\_password

This bypasses Gmail’s block and is secure.

**🔓 If You Don’t Use 2FA (Less Secure Option)**

1. Visit Less Secure App Access
2. Turn **ON** access for less secure apps
3. Use your regular Gmail password in .env

Google may disable this option soon, so App Passwords are preferred.

**🧪 Test Outside Streamlit (Optional)**

To confirm credentials work:

python

import smtplib

server = smtplib.SMTP("smtp.gmail.com", 587)

server.starttls()

server.login("jagdevsinghdosanjh@gmail.com", "your\_app\_password")

print("Login successful")

server.quit()

If this fails, the issue is with Gmail—not your app.

Once updated, restart your app:

powershell

streamlit run app.py

Let me know if you’d like help scaffolding a fallback SMS-only login or polishing .env.template for contributor clarity. You’re almost ready for classroom deployment 🌠

my streamlit secret key

Jagdev, if you're referring to the streamlit **secret key** used for securely storing credentials or API tokens in your app, here’s how to set it up properly using .streamlit/secrets.toml:

**✅ Create .streamlit/secrets.toml**

Inside your project root, create a folder named .streamlit and a file inside it called secrets.toml.

**📄 Example: .streamlit/secrets.toml**

toml

# Email credentials

EMAIL\_USER = "jagdevsinghdosanjh@gmail.com"

EMAIL\_PASS = "your\_app\_password\_here"

# Twilio credentials

TWILIO\_ACCOUNT\_SID = "your\_twilio\_sid"

TWILIO\_AUTH\_TOKEN = "your\_twilio\_token"

TWILIO\_PHONE\_NUMBER = "+91xxxxxxxxxx"

# OTP expiry

OTP\_EXPIRY\_MINUTES = 5

# Database path

DB\_PATH = "data/logs.db"

**🔐 Access Secrets in Code**

In your Python files, use:

python

import streamlit as st

email\_user = st.secrets["EMAIL\_USER"]

email\_pass = st.secrets["EMAIL\_PASS"]

This keeps sensitive data out of .env and makes your app more secure when deployed.

Would you like help converting your .env to .streamlit/secrets.toml, or scaffolding a config.toml to customize your app’s theme and splash visuals? Let’s make your constellation secure and beautiful 🌠

For working deployment on streamlit without any issue what is best.

For a smooth, secure, and scalable deployment of your Streamlit app—especially one as modular and classroom-facing as your OMR Evaluator—here’s a best-practice checklist tailored to your setup 🧠🚀✨

**✅ Streamlit Deployment Best Practices**

**1. Use .streamlit/secrets.toml for Credentials**

* Store sensitive keys like email, Twilio, and DB paths here.
* Avoid .env in production unless you're using a backend framework.

toml

EMAIL\_USER = "your\_email@gmail.com"

EMAIL\_PASS = "your\_app\_password"

TWILIO\_ACCOUNT\_SID = "..."

TWILIO\_AUTH\_TOKEN = "..."

DB\_PATH = "data/logs.db"

Access with:

python

import streamlit as st

email = st.secrets["EMAIL\_USER"]

**2. Use requirements.txt with Exact Versions**

Pin versions to avoid future breakage:

txt

streamlit==1.50.0

opencv-python==4.12.0.88

numpy==2.3.3

pandas==2.3.3

python-dotenv==1.1.1

bcrypt==5.0.0

email-validator==2.3.0

reportlab==4.4.4

Pillow==11.3.0

twilio==8.0.0

**3. Organize Your App Modularly**

Your current structure is excellent:

Code

📁 omr\_evaluator\_app/

├── app.py

├── pages/

├── modules/

├── utils/

├── assets/

├── data/

├── tests/

└── .streamlit/

**4. Use config.toml for Branding**

Inside .streamlit/config.toml:

toml

[theme]

primaryColor = "#6c63ff"

backgroundColor = "#f5f5f5"

secondaryBackgroundColor = "#ffffff"

textColor = "#333333"

font = "sans serif"

**5. Deploy with Streamlit Community Cloud or Docker**

**Option A: Streamlit Community Cloud**

* Push to GitHub
* Include:
  + requirements.txt
  + .streamlit/secrets.toml
  + app.py
* Go to streamlit.io/cloud and deploy

**Option B: Docker (for advanced deployment)**

Use a Dockerfile:

Dockerfile

FROM python:3.10

WORKDIR /app

COPY . .

RUN pip install -r requirements.txt

CMD ["streamlit", "run", "app.py", "--server.port=8501", "--server.address=0.0.0.0"]

**6. Test with streamlit run app.py Locally**

* Activate your virtual environment
* Run the app
* Check logs for errors
* Use tests/ to validate modules

Would you like me to scaffold a Dockerfile, polish your .streamlit/config.toml, or generate a secrets\_template.toml for contributors? You’re nearly ready for constellation-class deployment 🌠

How to get SECRET\_KEY=your\_streamlit\_secret\_key

Jagdev, great question. In Streamlit, SECRET\_KEY isn’t a required or standard setting like in Flask or Django. But if you're using it for encryption, session tokens, or custom security logic, here’s how to generate and use it securely:

**🔐 How to Generate a SECRET\_KEY**

You can generate a secure random key using Python:

python

import secrets

print(secrets.token\_hex(32))

This will output something like:

Code

e3f1c9a2b4d6e7f8a9c0d1e2f3a4b5c6d7e8f9a0b1c2d3e4f5a6b7c8d9e0f1a2

**✅ Where to Store It**

**Option A: .streamlit/secrets.toml**

toml

SECRET\_KEY = "e3f1c9a2b4d6e7f8a9c0d1e2f3a4b5c6d7e8f9a0b1c2d3e4f5a6b7c8d9e0f1a2"

Access it in your app:

python

import streamlit as st

secret\_key = st.secrets["SECRET\_KEY"]

**Option B: .env (if using python-dotenv)**

env

SECRET\_KEY=e3f1c9a2b4d6e7f8a9c0d1e2f3a4b5c6d7e8f9a0b1c2d3e4f5a6b7c8d9e0f1a2

Access it:

python

import os

from dotenv import load\_dotenv

load\_dotenv()

secret\_key = os.getenv("SECRET\_KEY")

**🧠 When to Use SECRET\_KEY**

Use it if your app includes:

* Token generation or encryption
* Custom session management
* Secure links or OTP verification
* Any logic that requires a consistent, secret seed

Would you like me to scaffold a security\_utils.py module to handle token generation, hashing, or encryption using your SECRET\_KEY? Or polish your secrets\_template.toml for contributor onboarding? Let’s lock it down with elegance 🔐✨

Check this

Jagdev, thanks for sharing the error snapshot from VS Code. The TOML syntax issues in your .streamlit/secrets.toml file are likely due to missing quotes, incorrect formatting, or misplaced characters. Let’s fix it cleanly so your Streamlit app runs without a hitch 🔧✨

**✅ Correct secrets.toml Format**

Here’s a clean, working example you can copy and adapt:

toml

EMAIL\_USER = "jagdevsinghdosanjh@gmail.com"

EMAIL\_PASS = "your\_app\_password\_here"

TWILIO\_ACCOUNT\_SID = "your\_twilio\_sid"

TWILIO\_AUTH\_TOKEN = "your\_twilio\_token"

TWILIO\_PHONE\_NUMBER = "+91xxxxxxxxxx"

OTP\_EXPIRY\_MINUTES = "5"

DB\_PATH = "data/logs.db"

SECRET\_KEY = "e3f1c9a2b4d6e7f8a9c0d1e2f3a4b5c6d7e8f9a0b1c2d3e4f5a6b7c8d9e0f1a2"

**🧠 Common TOML Mistakes to Avoid**

| **Mistake** | **Fix** |
| --- | --- |
| Missing quotes around strings | "value" instead of value |
| Mixing types (e.g. int + str) | Use consistent quoting or casting |
| Trailing commas or semicolons | TOML does **not** allow them |
| Misaligned indentation | TOML is flat—no nesting or tabs allowed |

**🔍 How to Validate**

You can use TOML Lint or VS Code’s “Even Better TOML” extension to check your file live.

Once fixed, restart your app:

powershell

streamlit run app.py

Let me know if you’d like to scaffold a secrets\_template.toml for contributors or polish your .streamlit/config.toml for splash visuals and theme alignment. You're almost ready for showcase deployment 🌠

Correct this EMAIL\_HOST=smtp.gmail.com EMAIL\_PORT=587 EMAIL\_USER=jagdevsinghdosanjh@gmail.com EMAIL\_PASS=jmyf yvql wnzv ooqe TWILIO\_ACCOUNT\_SID=AC89e2369884773b9ab95de90cc2814a1c TWILIO\_AUTH\_TOKEN=81a371d7c76b8ec19646194d6b798615 TWILIO\_PHONE\_NUMBER=+19093100618 SECRET\_KEY=5d282be71bd9f6cab4f010351129508999cc23985a0f761067e54eda8b208563 OTP\_EXPIRY\_MINUTES=5 DB\_PATH=data/logs.db

Here’s the corrected version of your configuration in **TOML format**, ready for .streamlit/secrets.toml. This ensures proper quoting and syntax so Streamlit can parse it without errors ✅

**✅ Corrected .streamlit/secrets.toml**

toml

EMAIL\_HOST = "smtp.gmail.com"

EMAIL\_PORT = "587"

EMAIL\_USER = "jagdevsinghdosanjh@gmail.com"

EMAIL\_PASS = "jmyf yvql wnzv ooqe"

TWILIO\_ACCOUNT\_SID = "AC89e2369884773b9ab95de90cc2814a1c"

TWILIO\_AUTH\_TOKEN = "81a371d7c76b8ec19646194d6b798615"

TWILIO\_PHONE\_NUMBER = "+19093100618"

SECRET\_KEY = "5d282be71bd9f6cab4f010351129508999cc23985a0f761067e54eda8b208563"

OTP\_EXPIRY\_MINUTES = "5"

DB\_PATH = "data/logs.db"

**🔍 Notes**

* All values are wrapped in **double quotes** "..." to avoid TOML parsing errors
* Even numeric values like EMAIL\_PORT and OTP\_EXPIRY\_MINUTES are quoted for safety
* This format is compatible with st.secrets["KEY\_NAME"] in Streamlit

Once saved, restart your app:

powershell

streamlit run app.py

Let me know if you'd like a secrets\_template.toml for contributors, or help validating access to Gmail and Twilio using these credentials. You're nearly ready for showcase deployment 🌠

git config --global user.name "jagdevsinghdosanjh"

git config --global user.email [jagdevsinghdosanjh@gmail.com](mailto:jagdevsinghdosanjh@gmail.com)