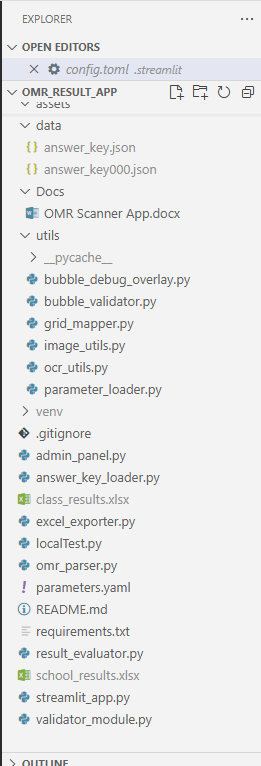
OMR\_RESULT\_APP



**Config.toml**

[server]

headless = true

enableCORS = false

port = 8501

**secrets\_template.toml**

# 🧠 OMR Result Generator - Secrets Template

# Rename this file to secrets.toml and fill in your actual credentials.

# Never commit secrets.toml to GitHub—it's ignored by .gitignore.

# 📤 Email dispatch (optional)

[email]

username = "jagdevsinghdosanjh@gmail.com"

password = "your\_email\_password"

smtp\_server = "smtp.gmail.com"

smtp\_port = 587

# 🗂️ External storage (optional)

[onedrive]

client\_id = "your\_onedrive\_client\_id"

client\_secret = "your\_onedrive\_client\_secret"

redirect\_uri = "your\_redirect\_uri"

# 🧪 OCR fallback path (Windows)

[tesseract]

path = "C:\\Program Files\\Tesseract-OCR\\tesseract.exe"

# 🏫 School-specific deployment (optional)

[school]

name = "Your School Name"

admin\_email = "admin@example.com"

district\_code = "ABC123"

# 🧑‍💻 Contributor onboarding

[contributor]

welcome\_message = "Welcome to the OMR Result Generator! Your modular clarity powers classroom celebration."

**settings.json**

{

    "python-envs.pythonProjects": []

}

**answer\_key.json**

{

  "Punjabi": {

    "answers": {

      "1": "2",

      "2": "2",

      "3": "2",

      "4": "1",

      "5": "3",

      "6": "3",

      "7": "3",

      "8": "3",

      "9": "3",

      "10": "3",

      "11": "3",

      "12": "4",

      "13": "1",

      "14": "3",

      "15": "1",

      "16": "4",

      "17": "3",

      "18": "3"

    }

  },

  "Math": {

    "answers": {

      "19": "2",

      "20": "3",

      "21": "2",

      "22": "2",

      "23": "1",

      "24": "1",

      "25": "2",

      "26": "2",

      "27": "3",

      "28": "2",

      "29": "1",

      "30": "4",

      "31": "1",

      "32": "1",

      "33": "1",

      "34": "3",

      "35": "3",

      "36": "1"

    }

  },

  "Science": {

    "answers": {

      "37": "2",

      "38": "2",

      "39": "3",

      "40": "3",

      "41": "1",

      "42": "1",

      "43": "2",

      "44": "2",

      "45": "3",

      "46": "3",

      "47": "3",

      "48": "4",

      "49": "4",

      "50": "2",

      "51": "3",

      "52": "4",

      "53": "2",

      "54": "3"

    }

  },

  "SST": {

    "answers": {

      "55": "3",

      "56": "4",

      "57": "1",

      "58": "3",

      "59": "2",

      "60": "4",

      "61": "3",

      "62": "3",

      "63": "1",

      "64": "1",

      "65": "2",

      "66": "2",

      "67": "2",

      "68": "4",

      "69": "4",

      "70": "3",

      "71": "1",

      "72": "3"

    }

  },

  "English": {

    "answers": {

      "73": "3",

      "74": "1",

      "75": "1",

      "76": "3",

      "77": "2",

      "78": "2",

      "79": "2",

      "80": "3",

      "81": "4",

      "82": "2",

      "83": "1",

      "84": "1",

      "85": "2",

      "86": "4",

      "87": "2",

      "88": "2",

      "89": "3",

      "90": "2"

    }

  },

  "Hindi": {

    "answers": {

      "91": "3",

      "92": "4",

      "93": "2",

      "94": "1",

      "95": "2",

      "96": "2",

      "97": "4",

      "98": "1",

      "99": "1",

      "100": "1",

      "101": "1",

      "102": "1",

      "103": "4",

      "104": "2",

      "105": "4",

      "106": "4",

      "107": "2",

      "108": "2"

    }

  }

}

**# utils/bubble\_debug\_overlay.py**

import cv2

def draw\_detected\_bubbles(img, bubbles, missing\_count=0):

    """

    Draws circles around detected bubbles and overlays a poetic message.

    `bubbles` should be a list of (x, y, r) tuples.

    """

    overlay = img.copy()

    for (x, y, r) in bubbles:

        cv2.circle(overlay, (x, y), r, (0, 255, 0), 2)

    if missing\_count > 0:

        poetic\_text = f"{missing\_count} bubbles drifted into silence.\nLet clarity guide the next scan."

        cv2.putText(overlay, poetic\_text, (30, 50), cv2.FONT\_HERSHEY\_SIMPLEX, 0.6, (0, 0, 255), 2)

    return overlay

**# utils/bubble\_validator.py**

import cv2

def validate\_bubbles(bubbles, expected\_count=None, actual\_count=None):

    malformed = [b for b in bubbles if len(b) != 3]

    valid = [b for b in bubbles if len(b) == 3]

    summary = {

        "valid": len(valid),

        "malformed": len(malformed),

        "missing": expected\_count - actual\_count if expected\_count and actual\_count is not None else None

    }

    return summary

def draw\_validation\_overlay(img, bubbles, summary):

    overlay = img.copy()

    for b in bubbles:

        if len(b) == 3:

            x, y, r = b

            cv2.circle(overlay, (x, y), r, (0, 255, 0), 2)

        else:

            x, y = b[:2] if len(b) >= 2 else (30, 30)

            cv2.circle(overlay, (x, y), 10, (0, 0, 255), 2)

    if summary.get("missing", 0) > 0:

        poetic\_text = f"{summary['missing']} bubbles drifted into silence.\nLet clarity guide the next scan."

        cv2.putText(overlay, poetic\_text, (30, 50), cv2.FONT\_HERSHEY\_SIMPLEX, 0.6, (255, 255, 255), 2)

    return overlay

# utils/grid\_mapper.py

def generate\_question\_grid(start\_x, start\_y, dx, dy, rows, cols):

    """

    Generates a grid of bubble centers for mapping.

    Each row = one question, each col = one option (A–D).

    """

    grid = {}

    for q in range(rows):

        grid[q] = []

        for o in range(cols):

            x = start\_x + o \* dx

            y = start\_y + q \* dy

            grid[q].append((x, y))  # Option A–D

    return grid

**# utils/image\_utils.py**

import cv2

import numpy as np  # noqa

def preprocess\_image(img):

    """

    Converts image to grayscale, applies Gaussian blur, and thresholds using Otsu's method.

    Returns the original image and binary thresholded image.

    """

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

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

    thresh = cv2.threshold(blurred, 0, 255, cv2.THRESH\_BINARY\_INV + cv2.THRESH\_OTSU)[1]

    return img, thresh

def detect\_bubbles(thresh):

    """

    Detects circular contours from a thresholded image.

    Returns a list of (x, y, r) tuples representing bubble centers and radii.

    """

    bubbles = []

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

    for cnt in contours:

        if len(cnt) >= 5:  # Ensure contour is valid for enclosing circle

            (x, y), radius = cv2.minEnclosingCircle(cnt)

            if 5 < radius < 20:  # Filter by size

                bubbles.append((int(x), int(y), int(radius)))

    return bubbles

def map\_bubbles\_to\_responses(bubbles, question\_grid, radius\_tolerance=15):

    """

    Maps detected bubbles to question options using proximity to grid centers.

    Returns a dictionary of responses: {question\_number: selected\_option\_letter}.

    """

    responses = {}

    for q, options in question\_grid.items():

        for idx, (x\_ref, y\_ref) in enumerate(options):

            for bubble in bubbles:

                if len(bubble) == 3:

                    x, y, r = bubble

                    dist = ((x - x\_ref)\*\*2 + (y - y\_ref)\*\*2)\*\*0.5

                    if dist < radius\_tolerance:

                        responses[q] = chr(65 + idx)  # A, B, C, D

                        break  # Stop after first match

                else:

                    print(f"⚠️ Skipping malformed bubble: {bubble}")

    return responses

**ocr\_utils.py**

# OCR logic for metadata extraction

import pytesseract

# Explicit fallback path for Windows systems

pytesseract.pytesseract.tesseract\_cmd = r"C:\Program Files\Tesseract-OCR\tesseract.exe"

def extract\_metadata(img):

    """

    Extracts student name and roll number from a decoded OpenCV image.

    Returns a dictionary with metadata.

    """

    text = pytesseract.image\_to\_string(img)

    name = roll\_no = ""

    for line in text.split("\n"):

        if "Name" in line:

            name = line.split(":")[-1].strip()

        elif "Roll" in line:

            roll\_no = line.split(":")[-1].strip()

    return {"name": name, "roll\_no": roll\_no}

**parameter\_loader.py**

import yaml

import streamlit as st

def load\_parameters(uploaded\_file=None):

    if uploaded\_file:

        return yaml.safe\_load(uploaded\_file)

    try:

        with open("parameters.yaml", "r", encoding="utf-8") as f:  # ✅ Force UTF-8

            return yaml.safe\_load(f)

    except FileNotFoundError:

        st.warning("⚠️ parameters.yaml not found. Using defaults.")

    except UnicodeDecodeError:

        st.error("❌ parameters.yaml contains invalid characters. Please save it as UTF-8.")

    return {

        "school\_name": "Unnamed School",

        "grading\_scheme": "standard",

        "leaderboard\_limit": 10

    }

**admin\_panel.py**

# School-wide dashboard and export

import streamlit as st

import pandas as pd

from excel\_exporter import export\_to\_excel

def launch\_admin\_panel(results, theme="constellation"):

    # Use theme to customize visuals, badge overlays, etc.

    st.subheader("🏫 School-Wide Dashboard")

# def launch\_admin\_panel(results):

#     st.header("🏫 School-Wide Dashboard")

    df = pd.DataFrame(results)

    st.dataframe(df)

    st.subheader("📊 Leaderboard")

    top\_students = df.sort\_values(by="Score", ascending=False).head(10)

    st.table(top\_students[["Name", "Roll No", "Score"]])

    st.subheader("📤 Export Full School Results")

    export\_to\_excel(results, "school\_results.xlsx")

    with open("school\_results.xlsx", "rb") as f:

        st.download\_button("Download School Results", f, file\_name="school\_results.xlsx")

**answer\_key\_loader.py**

# Loads and flattens answer key JSON

import json

def load\_answer\_key(path="data/answer\_key.json") -> dict:

    with open(path, "r") as f:

        raw = json.load(f)

    flat\_key = {}

    for subject\_block in raw.values():

        flat\_key.update(subject\_block["answers"])

    return {int(k): int(v) for k, v in flat\_key.items()}

**excel\_exporter.py**

# Exports results to Excel

import pandas as pd

def export\_to\_excel(results, filename="class\_results.xlsx"):

    df = pd.DataFrame(results)

    df.to\_excel(filename, index=False)

**omr\_parser.py**

from utils.grid\_mapper import generate\_question\_grid

from utils.image\_utils import detect\_bubbles, map\_bubbles\_to\_responses

from utils.image\_utils import preprocess\_image

def extract\_responses(img):

    \_, thresh = preprocess\_image(img)

    bubbles = detect\_bubbles(thresh)

    question\_grid = generate\_question\_grid(

        start\_x=100, start\_y=200, dx=40, dy=30,

        rows=50, cols=4

    )

    responses = map\_bubbles\_to\_responses(bubbles, question\_grid)

    return {"responses": responses, "bubbles": bubbles}

**parameters.yaml**

# 🧠 School Identity

school\_name: "Government High School Chananke"

admin\_email: "jagdevsinghdosanjh@gmail.com"

# 🎓 Grading Configuration

grading\_scheme: "weighted"  # Options: standard, weighted, adaptive

leaderboard\_limit: 10       # Number of top scorers to display

# 🧮 Subject Weights (used if grading\_scheme is 'weighted')

subject\_weights:

  Punjabi: 1.0

  Math: 1.5

  Science: 1.2

  SST: 1.0

  English: 1.3

  General: 0.8

# 📊 Question Configuration

expected\_questions: 108

subject\_question\_counts:

  Punjabi: 18

  Math: 18

  Science: 18

  SST: 18

  English: 18

  General: 18

# 🏆 Badge Celebration Themes

badge\_theme: "constellation"  # Options: constellation, ripple, legacy, remix

# 📦 Export Settings

export\_filename: "class\_results.xlsx"

include\_timestamp: true

# 🧑‍🏫 Admin Panel Modules

enable\_feedback\_module: true

enable\_remix\_tracker: true

enable\_badge\_recommender: true

**result\_evaluator.py**

# Compares student responses and scores

def evaluate\_responses(student\_responses, answer\_key):

    score = 0

    correct = {}

    incorrect = {}

    for q\_no, selected in student\_responses.items():

        correct\_ans = answer\_key.get(q\_no)

        if selected == correct\_ans:

            score += 1

            correct[q\_no] = selected

        else:

            incorrect[q\_no] = selected

    return {

        "score": score,

        "correct": correct,

        "incorrect": incorrect

    }

**streamlit\_app.py**

import streamlit as st

import numpy as np

import cv2

from utils.parameter\_loader import load\_parameters

from omr\_parser import extract\_responses

from answer\_key\_loader import load\_answer\_key

from result\_evaluator import evaluate\_responses

from excel\_exporter import export\_to\_excel

from validator\_module import validate\_sheet, poetic\_feedback

from admin\_panel import launch\_admin\_panel

from utils.ocr\_utils import extract\_metadata

from utils.bubble\_validator import validate\_bubbles, draw\_validation\_overlay

# 🌟 Page setup

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

st.title("📄 OMR Result Generator")

# 📂 Sidebar Uploads

with st.sidebar:

    uploaded\_params = st.file\_uploader("Upload parameters.yaml", type=["yaml", "yml"])

    uploaded\_files = st.file\_uploader("Upload OMR Sheets", type=["jpg", "jpeg", "png"], accept\_multiple\_files=True)

# 🧠 Load Parameters

if uploaded\_params:

    parameters = load\_parameters(uploaded\_params)

    st.success("✅ parameters.yaml loaded.")

else:

    st.warning("⚠️ Please upload parameters.yaml.")

    parameters = load\_parameters(None)

# 🌈 Dynamic Splash Screen

st.markdown(f"### 🌟 Welcome to {parameters['school\_name']} OMR Showcase!")

# 🧮 Load Answer Key

answer\_key = load\_answer\_key()

# 📊 Process Uploaded Sheets

if uploaded\_files:

    results = []

    expected = parameters.get("expected\_questions", 108)

    for file in uploaded\_files:

        file.seek(0)

        file\_bytes = np.asarray(bytearray(file.read()), dtype=np.uint8)

        img = cv2.imdecode(file\_bytes, cv2.IMREAD\_COLOR)

        if img is None:

            st.error(f"❌ Failed to decode image: {file.name}")

            continue

        extracted = extract\_responses(img)

        metadata = extract\_metadata(img)

        responses = extracted["responses"]

        bubbles = extracted.get("bubbles", [])

        actual = len(responses)

        missing = expected - actual

        # 🧪 Bubble Validation Overlay

        summary = validate\_bubbles(bubbles, expected\_count=expected, actual\_count=actual)

        debug\_img = draw\_validation\_overlay(img.copy(), bubbles, summary)

        st.image(debug\_img, caption=f"🧪 Bubble Validation Overlay for {metadata['name']}")

        # 🎓 Student Header

        st.markdown(f"#### 🎓 Student: {metadata['name']} | Roll No: {metadata['roll\_no']}")

        # 🌈 Poetic Feedback

        if missing > 0:

            st.markdown(f"""

            > ✨ \*Some bubbles wandered, some stayed shy —

            > Let’s guide them gently, before they fly.\*

            > \*\*Missing responses:\*\* {missing}

            """)

        else:

            st.success("🌟 All responses captured with clarity!")

        # 🧠 Validation + Feedback

        issues = validate\_sheet(responses)

        poetic\_feedback(issues)

        # 🧮 Evaluation

        evaluation = evaluate\_responses(responses, answer\_key)

        result = {

            "Name": metadata["name"],

            "Roll No": metadata["roll\_no"],

            "Score": evaluation["score"]

        }

        results.append(result)

    if results:

        st.success(f"✅ Processed {len(results)} students")

        export\_to\_excel(results)

        with open("class\_results.xlsx", "rb") as f:

            st.download\_button("📥 Download Excel", f, file\_name=parameters.get("export\_filename", "class\_results.xlsx"))

        # 🧑‍🏫 Launch Admin Panel

        launch\_admin\_panel(results)

**validator\_module.py**

# Flags incomplete or ambiguous sheets

import streamlit as st

def validate\_sheet(responses, expected\_questions=90):

    issues = []

    if len(responses) < expected\_questions:

        issues.append(f"🕳️ Missing responses: {expected\_questions - len(responses)} questions unanswered.")

    duplicates = [q for q, v in responses.items() if isinstance(v, list) and len(v) > 1]

    if duplicates:

        issues.append(f"🔁 Multiple bubbles detected in: {duplicates}")

    return issues

def poetic\_feedback(issues):

    if not issues:

        return "✨ All bubbles aligned. The sheet sings in clarity."

    st.markdown("> \_“Some bubbles wandered, some stayed shy—\nLet’s guide them gently, before they fly.”\_")

    for issue in issues:

        st.warning(issue)