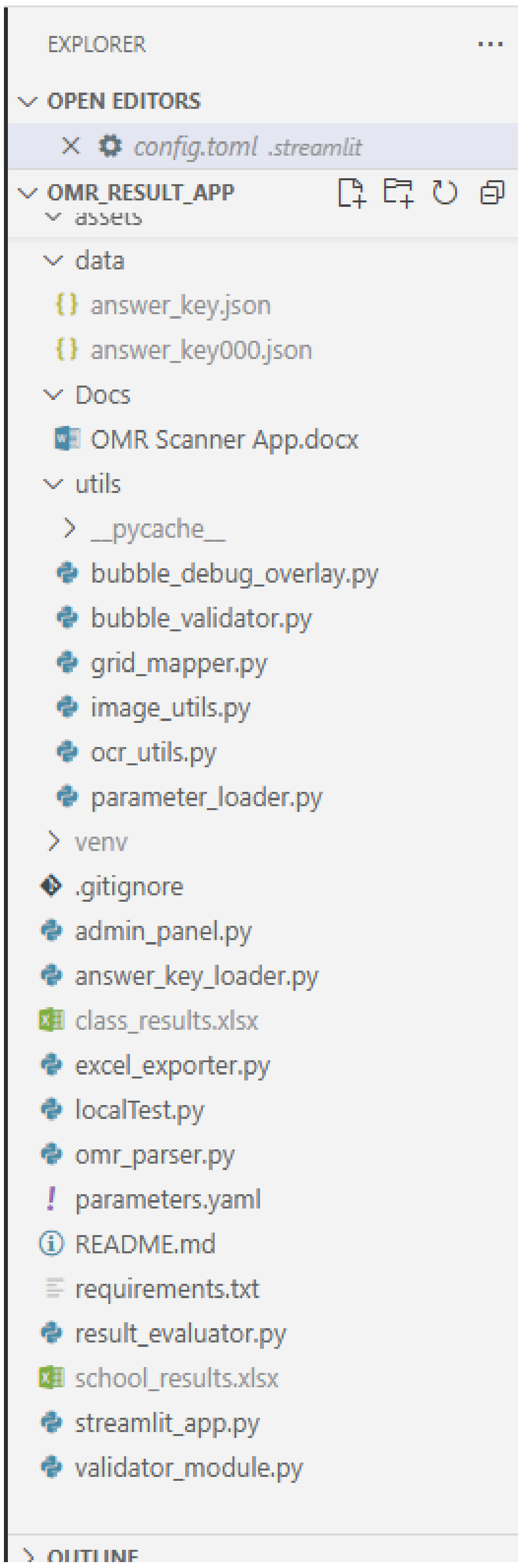
OMR_RESULT_APP



```
EXPLORER ...
∨ OPEN EDITORS
    × config.toml .streamlit
∨ OMR_RESULT_APP
  ∨ assets
  ∨ data
      answer_key.json
      answer_key000.json
  ∨ Docs
      OMR Scanner App.docx
  ∨ utils
    > __pycache__
      bubble_debug_overlay.py
      bubble_validator.py
      grid_mapper.py
      image_utils.py
      ocr_utils.py
      parameter_loader.py
  > venv
    .gitignore
    admin_panel.py
    answer_key_loader.py
    class_results.xlsx
    excel_exporter.py
    localTest.py
    omr_parser.py
    parameters.yaml
    README.md
    requirements.txt
    result_evaluator.py
    school_results.xlsx
    streamlit_app.py
    validator_module.py
> OUTLINE
```

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

```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

```python
# 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)
```