

COSC3506 FINAL PROJECT

STUDENT GRADE TRACKER

Muhammad Jamil

1. General Overview of the Software Project

The Student Grade Tracker is a full-stack software solution built to simplify the academic evaluation workflow. It supports grade input, visualization, and export functionalities. Hosted on Google Colab, this application is ideal for educators who need a lightweight and accessible performance management system.

It was developed to demonstrate software engineering best practices, and it runs entirely on Python-based tools with no need for local setup. This makes it an excellent candidate for real-time demos and academic stakeholder presentations.

2. Project Scope

The software enables seamless academic tracking in a lightweight, cloud-accessible environment. It is modular and scalable, capable of being extended into full LMS integration.

3. Project Capabilities

Functional Requirements:

- Teachers can add, update, and delete student grades for any subject.
- Users can view all grades for a student, along with GPA and percentage average.
- Grades are visualized as bar charts.
- Grade data can be exported to CSV format for reports.
- New students and subjects can be added dynamically.

SOME SCREENSHOTS(GRADING CAPABILITIES):

+ Add / Update Grade

Student Username

Muhammad Jamil

Subject

Computer

Grade

79

↺

0

100

✓ Add / Update

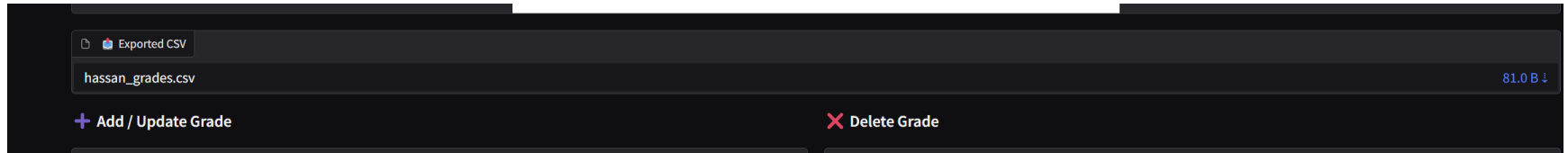
Status

Grade updated.

GRADE VISUALIZATION:



CSV DEMO SCREENSHOT



Non-Functional Requirements:

- The application must run in Google Colab without requiring local installation.
- Response times must be under 2 seconds for all operations.
- UI must be visually intuitive and accessible.

User Roles:

- **Teacher/Admin:** Can manage all student data.
- **Student/Viewer:** Can view their data and performance.

UNDER 2 SECONDS RESPONSE TIME(NON FUNCTIONAL REQ DEMO):

Student Username

Muhammad Jamil

Subject

Chemistry

Grade

94

↺

0

100

✓ Add / Update

Status

0.4s

4. Business Value Provided

- Automates academic record keeping for instructors.
- Offers students immediate feedback through visualized GPA and grades.
- Reduces paperwork and errors via automated calculations and exports.
- Ready to scale into larger educational systems or LMS tools.
- Demonstrates practical use of software engineering and UI/UX skills for investors.

5. Software Engineering Model Chosen and Why

We implemented the Agile methodology using the DoFARE framework. DoFARE stands for:

1. **Define** - Establishing a clear vision and scope for each module.
2. **Organize** - Structuring backend, frontend, and database components.
3. **Formalize** - Defining APIs and I/O expectations with function specs.
4. **Analyze** - Testing functionality through real data inputs and outputs.
5. **Review** - Peer and self-review of components during iterations.

6. **Execute** - Final integration and deployment to Colab.

This model was chosen because the project was being developed iteratively over two months, and DoFARE offers an excellent balance of flexibility and formalism. Its structure allowed us to work in focused stages, review our progress regularly, and deliver working components at each milestone.

Why Not Other Models? We considered other models like Waterfall and Spiral.

Waterfall, while structured, would not have supported our need for ongoing iteration and early testing. Once a stage is complete in Waterfall, going back is difficult — a risk we couldn't afford for a demo-heavy project.

The Spiral model emphasizes risk analysis, which was not our primary concern. It is more suited for mission-critical systems, whereas our focus was usability, UI responsiveness, and quick iteration.

Agile with DoFARE fit perfectly for an academic software project — especially when feedback cycles, evolving UI requirements, and frequent updates were necessary.

6. Tools, Programming Languages, and Frameworks Used

Component	Technology Used
------------------	------------------------

Frontend:	Gradio (Python-based UI)
------------------	--------------------------

Backend:	Flask
-----------------	-------

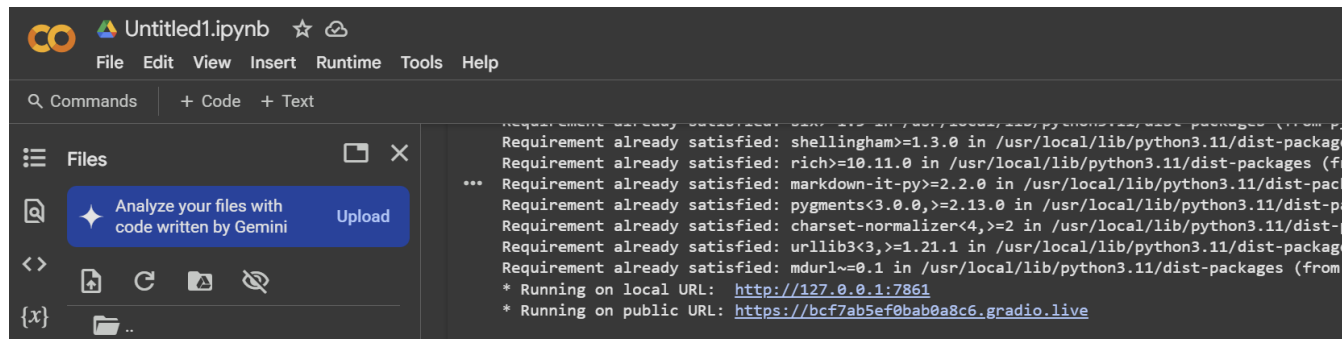
Database:	SQLite
------------------	--------

Data Export:	Pandas
---------------------	--------

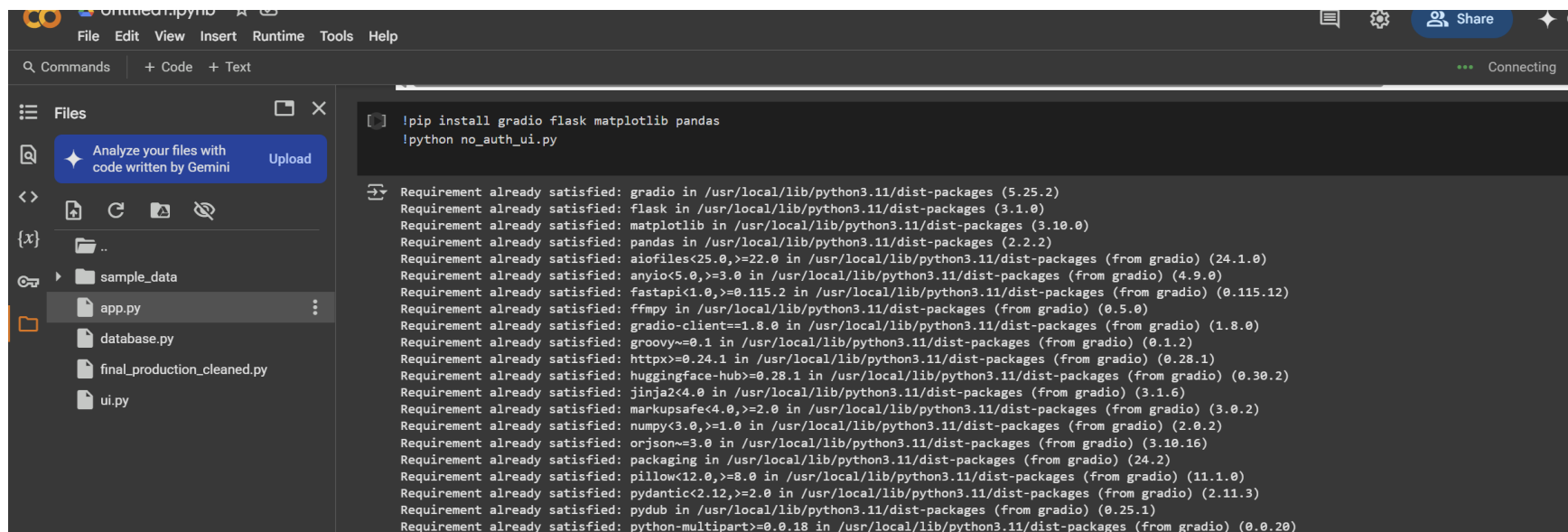
Graphs:	Matplotlib
----------------	------------

Hosting:	Google Colab
-----------------	--------------

Hosting Screenshot collab:



```
Requirement already satisfied: shellingham>=1.3.0 in /usr/local/lib/python3.11/dist-packages (from py)
Requirement already satisfied: rich>=10.11.0 in /usr/local/lib/python3.11/dist-packages (from py)
... Requirement already satisfied: markdown-it-py>=2.2.0 in /usr/local/lib/python3.11/dist-packa
Requirement already satisfied: pygments<3.0.0,>=2.13.0 in /usr/local/lib/python3.11/dist-pac
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.11/dist-pa
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.11/dist-packages
Requirement already satisfied: mdurl~=0.1 in /usr/local/lib/python3.11/dist-packages (from m
* Running on local URL: http://127.0.0.1:7861
* Running on public URL: https://bcf7ab5ef0bab0a8c6.gradio.live
```



```
!pip install gradio flask matplotlib pandas
!python no_auth_ui.py

Requirement already satisfied: gradio in /usr/local/lib/python3.11/dist-packages (5.25.2)
Requirement already satisfied: flask in /usr/local/lib/python3.11/dist-packages (3.1.0)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.11/dist-packages (3.10.0)
Requirement already satisfied: pandas in /usr/local/lib/python3.11/dist-packages (2.2.2)
Requirement already satisfied: aiofiles<25.0,>=22.0 in /usr/local/lib/python3.11/dist-packages (from gradio) (24.1.0)
Requirement already satisfied: anyio<5.0,>=3.0 in /usr/local/lib/python3.11/dist-packages (from gradio) (4.9.0)
Requirement already satisfied: fastapi<1.0,>=0.115.2 in /usr/local/lib/python3.11/dist-packages (from gradio) (0.115.12)
Requirement already satisfied: ffmpeg in /usr/local/lib/python3.11/dist-packages (from gradio) (0.5.0)
Requirement already satisfied: gradio-client==1.8.0 in /usr/local/lib/python3.11/dist-packages (from gradio) (1.8.0)
Requirement already satisfied: groovy~=0.1 in /usr/local/lib/python3.11/dist-packages (from gradio) (0.1.2)
Requirement already satisfied: httpx>=0.24.1 in /usr/local/lib/python3.11/dist-packages (from gradio) (0.28.1)
Requirement already satisfied: huggingface-hub>=0.28.1 in /usr/local/lib/python3.11/dist-packages (from gradio) (0.30.2)
Requirement already satisfied: Jinja2<4.0 in /usr/local/lib/python3.11/dist-packages (from gradio) (3.1.6)
Requirement already satisfied: MarkupSafe<4.0,>=2.0 in /usr/local/lib/python3.11/dist-packages (from gradio) (3.0.2)
Requirement already satisfied: numpy<3.0,>=1.0 in /usr/local/lib/python3.11/dist-packages (from gradio) (2.0.2)
Requirement already satisfied: orjson~=3.0 in /usr/local/lib/python3.11/dist-packages (from gradio) (3.10.16)
Requirement already satisfied: packaging in /usr/local/lib/python3.11/dist-packages (from gradio) (24.2)
Requirement already satisfied: pillow<12.0,>=8.0 in /usr/local/lib/python3.11/dist-packages (from gradio) (11.1.0)
Requirement already satisfied: pydantic<2.12,>=2.0 in /usr/local/lib/python3.11/dist-packages (from gradio) (2.11.3)
Requirement already satisfied: pydub in /usr/local/lib/python3.11/dist-packages (from gradio) (0.25.1)
Requirement already satisfied: python-multipart>=0.0.18 in /usr/local/lib/python3.11/dist-packages (from gradio) (0.0.20)
```

7. Code Architecture & Explanation

Database Initialization (database.py)

def init_db():

```
conn = sqlite3.connect("grades.db")  
cur = conn.cursor()  
cur.execute("CREATE TABLE IF NOT EXISTS grades (  
    id INTEGER PRIMARY KEY,  
    student TEXT,  
    subject TEXT,  
    grade REAL,  
    timestamp TEXT  
    )")
```

This function initializes the SQLite database and creates a table grades with schema to track students, subjects, and scores.

Add/Update Grade Logic

```
def add_grade(student, subject, grade):  
    conn = sqlite3.connect("grades.db")  
    cur = conn.cursor()
```

```
cur.execute("SELECT id FROM grades WHERE student=? AND subject=?",  
(student, subject))  
  
if cur.fetchone():  
  
    cur.execute("UPDATE grades SET grade=? WHERE student=? AND  
subject=?", (grade, student, subject))  
  
    return "Grade updated."  
  
else:  
  
    cur.execute("INSERT INTO grades (student, subject, grade) VALUES (?, ?,  
?)", (student, subject, grade))  
  
    return "Grade added."
```

This implements an upsert pattern using SQLite. It first checks if the grade exists for a student-subject pair. If it does, it updates the value; otherwise, it inserts a new entry.

Gradio UI Snippet (production UI)

```
grade_slider = gr.Slider(0, 100, label="Grade", step=1)  
  
add_button.click(fn=add_student_grade, inputs=[add_student, add_subject,  
grade_slider], outputs=add_status)
```

This slider allows numeric grade input, and the click handler ensures the input is passed to the backend properly using Gradio's reactive bindings.

8. Student Role and Contribution

Muhammad Jamil

Note: My main role was Backend. But I have designed frontend as well.

- Designed the database schema and implemented CRUD operations using SQLite.
- Built and styled the user interface using Gradio blocks with accessibility and dark theme in mind.
- Implemented GPA and average logic based on percentage-to-GPA conversion.
- Added CSV export logic using Pandas and integrated it into the frontend.
- Fixed bugs related to type conflicts and enhanced error messaging.
- Ensured all components follow the DoFARE methodology and agile sprints.
- Led integration of frontend, backend, and visualization components for final production-ready version.

9. Conclusion

The Student Grade Tracker is a compact but powerful demonstration of applying software engineering principles to build a real-world usable tool. It follows structured planning, agile delivery, and clean code practices. The use of DoFARE model ensured that each feature was delivered with purpose and polish, making it suitable for both academic credit and professional demonstration.