CODING CHALLENGE

Ref: FP-10-30-25-001

Student Fee Management System

1. **Duration**: 6 Hours
2. **Stack**: Python or JavaScript
3. **Type**: Full Practical Project
4. **Difficulty**: Meduim

Problem Context

Uganda Christian University wants to computerize how it manages student fees. Your task is to develop a **Student Fee Management System** that can handle:

- ➔ Student registration
- ➔ Fee structure setup
- ➔ Payment recording
- ➔ Balance computation
- ➔ Summary reporting

The system should allow administrators to track payments efficiently and view total income and outstanding balances per program or campus.

System Requirements

You must build a program (CLI or web-based) that supports the following features:

1. Student Management

- ➔ Add new students with:
  - ◆ student_id
  - ◆ name
  - ◆ program
  - ◆ campus
  - ◆ year_of_study
- ➔ View all registered students.
- ➔ Edit or delete a student record.

2. Fee Structure Management

- ➔ Define a fee structure per program, e.g.:
  { "program": "BSc Computer Science", "year": 1, "total_fee": 2_000_000 }

- ➔ Allow updates to the fee amount.
- ➔ Display all fee structures.

3. **Payment Recording**

- ➔ Record payments made by students. Each payment record should have:
    - ◆ payment_id
    - ◆ student_id
    - ◆ amount
    - ◆ date
- ➔ Validate that total payments do not exceed the required fee.

4. **Balance Computation**

- ➔ For each student:
    - ◆ Compute total amount paid.
    - ◆ Compute balance = total_fee - total_paid.
- ➔ Mark students as **"Cleared"** if balance = 0, otherwise **"Not Cleared"**.

5. **Reports**

Generate reports such as:

- ➔ **Per Student** — Name, Program, Campus, Total Fee, Paid, Balance, Status.
- ➔ **Per Program** — Total expected income, total collected, outstanding balance.
- ➔ **Overall Summary** — Total collected across all programs.

6. **File Storage (Optional for CLI)**

- ➔ Store all data in JSON or CSV files to maintain persistence between runs. Example files: students.json, payments.json, fees.json.

7. **Bonus**

- ➔ Implement search and filtering (by program, campus, or payment status).
- ➔ Export reports to a CSV or text file.
- ➔ Handle user authentication (e.g., admin login).
- ➔ For web version (JS): implement a small REST API or browser interface.

8. Validation Guide (100 points)

Category	Description	Points
Core Functionality (40)	Student registration, fee setup, payment recording, and balance calculation	40
Data Management (15)	Correct data storage, validation, and retrieval	15
Report Generation (15)	Accurate and clear reports per student and program	15
Code Quality (10)	Modularity, use of functions/classes, readability, comments	10
Error Handling & Validation (10)	Input validation, prevention of invalid data	10
Bonus Features (10)	Search, filters, exports, or authentication	10

Implementation Hints

If using Python:

- ➔ Use classes: Student, Payment, FeeStructure, and FeeTracker.
- ➔ Use json module for file handling.
- ➔ Optionally use prettytable for neat CLI tables.

If using JavaScript:

- ➔ Use class syntax or modular functions.
- ➔ Use Node.js for CLI (readline, fs for file handling).
- ➔ Optionally use Express.js + JSON files or a simple frontend UI.

Good to have

- ➔ Add login (admin vs. normal user).
- ➔ Implement REST API endpoints (if in JS):
    - ◆ POST /students, POST /payments, GET /report

- ➔ Add totals grouped by **campus** and **program** dynamically.
- ➔ Integrate with CSV export (using Python csv or JS csv-writer).