PROJET D'EXAMEN · MULTI-LANGAGES

Projet 8
Système de notes scolaires

Bulletins, classements et statistiques de classe

Domaine :	Éducation · Évaluation · Statistiques
Composantes :	C (DLL) · Python (module + tests) · Flask (API + dashboard) · C# (GUI)
Cours :	5e TTR Informatique · 6e TTR Informatique
Format :	Projet individuel
Édition :	19 mai 2026

1 Contexte et objectifs

Tu construis un système de gestion de notes pour une école. Tu enregistres des élèves, des cours, des évaluations, et l'application calcule des moyennes pondérées par cours, des moyennes générales, des classements de classe et des distributions de notes. Le bulletin de chaque élève est imprimable depuis l'interface C#.

Objectifs pédagogiques

- Compiler une bibliothèque C en DLL et l'appeler depuis Python via `ctypes`.
- Écrire un module Python testable, avec validation et persistance des données.
- Concevoir une API REST Flask et un dashboard Chart.js qui la consomme.
- Développer une interface C# Windows Forms qui communique avec l'API.
- Comparer trois approches de calcul (Python pur, ctypes scalaire, ctypes batch).

2 Cahier des charges fonctionnel

L'application doit permettre à l'utilisateur de :

- CRUD élèves (nom, classe).
- CRUD cours (nom, professeur, coefficient).
- Saisie de notes (élève × évaluation, valeur sur 20, pondération).
- Moyenne pondérée par élève par cours.
- Moyenne générale par élève (cours pondérés par leur coefficient).
- Classement de classe par cours et général.
- Distribution des notes (histogramme) par cours.
- Bulletin imprimable par élève (PDF ou affichage formaté).

> **Moyenne pondérée —** $\mu = \Sigma(value_i \times weight_i) / \Sigma(weight_i)$

> **Catégories d'appréciation —** ≥ 16 : excellent · 14-16 : très bien · 12-14 : bien · 10-12 : suffisant · < 10 : insuffisant

3 Module C — `calculs.c` → `calculs.dll`

Les fonctions ci-dessous sont à exporter dans la DLL. Chaque fonction doit être présente en version **scalaire** (pour l'application normale) ; les fonctions marquées *batch* servent au benchmark de la Partie 5.

Signature	Rôle
`float weighted_avg(float* values, float* weights, int n);`	Moyenne pondérée
`float std_deviation(float* values, int n);`	Écart-type
`float percentile(float* values, int n, int p);`	Centile, p ∈ [0, 100]
`float min_max_normalize(float v, float min, float max);`	Échelle [0, 1]
`void weighted_avg_batch(float** vals, float** wts, int* ns, float* out, int n);`	Batch moyennes pondérées

Compilation Windows : `gcc -shared -o calculs.dll calculs.c -lm`

4 Module Python — `module.py`

Le module Python est le seul pont entre la DLL et le reste de l'application. Il contient les wrappers `ctypes` et la logique métier (validation, persistance, statistiques).

- `weighted_avg(values, weights) -> float — wrapper`
- `std_deviation(values) -> float — wrapper`
- `percentile(values, p) -> float — wrapper`
- `add_student(name, class_), list_students(), get_student(id)`
- `add_course(name, teacher, coeff), list_courses()`
- `add_grade(student_id, course_id, value, weight, title)`
- `student_average(student_id, course_id) -> float — moyenne pondérée du cours`
- `student_general_average(student_id) -> float — moyenne générale`
- `class_ranking(course_id) -> list[dict] — élèves triés par moyenne`
- `grade_distribution(course_id) -> dict — buckets de 2 pts`

5 API Flask — `app.py`

Toutes les réponses respectent le format `{"status": "ok|error", "data": ...}`. Statuts HTTP : 200 (lecture), 201 (création), 400 (validation), 404 (non trouvé), 500 (erreur serveur).

Méthode	Chemin	Description
GET	`/api/students`	Liste
POST	`/api/students`	Création
GET	`/api/students/{id}/bulletin`	Bulletin complet (toutes moyennes)
GET	`/api/courses`	Liste
POST	`/api/courses`	Création
GET	`/api/grades?student_id=&course_id=`	Filtré
POST	`/api/grades`	Ajout
GET	`/api/courses/{id}/ranking`	Classement classe
GET	`/api/courses/{id}/distribution`	Distribution histogramme

6 Dashboard web — `/dashboard`

Dashboard lecture seule, rafraîchissement automatique toutes les 30 secondes via fetch JS. Trois graphiques Chart.js obligatoires :

1. Moyennes par cours (bar chart).
2. Distribution des notes pour un cours sélectionné (histogram).
3. Évolution de la moyenne d'un élève sur l'année (line chart).

7 Interface C# — Windows Forms

Application Windows Forms qui communique avec l'API uniquement via `HttpClient`. Aucun calcul ne doit être fait côté C#.

- StudentsForm — gestion élèves.
- CoursesForm — gestion cours.
- GradeEntryForm — grille élèves × évaluations (DataGridView éditable).
- BulletinForm — bulletin formaté par élève avec moyennes et appréciations.
- ClassRankingForm — classement par cours sélectionné.

8 Format des données

Variante 5e — Persistance JSON

```
{
  "student": { "id": 1, "name": "Alice Dupont", "class": "5TTR" },
  "course":  { "id": 2, "name": "Mathématiques", "teacher": "M. Dupont", "coeff": 4 },
  "grade":   {
    "id": 17, "student_id": 1, "course_id": 2,
    "value": 15.5, "weight": 2.0,
    "title": "Interro chapitre 3",
    "date": "2026-05-15"
  }
}
```

Variante 6e — Persistance SQL (MySQL/SQLite)

```
CREATE TABLE students (
  id    INTEGER PRIMARY KEY AUTOINCREMENT,
  name  TEXT NOT NULL,
  class TEXT NOT NULL
);
CREATE TABLE courses (
  id      INTEGER PRIMARY KEY AUTOINCREMENT,
  name    TEXT NOT NULL,
  teacher TEXT NOT NULL,
  coeff   INTEGER NOT NULL DEFAULT 1
);
CREATE TABLE grades (
  id         INTEGER PRIMARY KEY AUTOINCREMENT,
  student_id INTEGER NOT NULL REFERENCES students(id),
  course_id  INTEGER NOT NULL REFERENCES courses(id),
  value      REAL NOT NULL CHECK (value BETWEEN 0 AND 20),
  weight     REAL NOT NULL DEFAULT 1.0,
  title      TEXT,
  date       TEXT NOT NULL DEFAULT CURRENT_DATE
);
CREATE INDEX idx_grades_student ON grades(student_id);
CREATE INDEX idx_grades_course  ON grades(course_id);
```

9 Plan de tests Pytest

Minimum 10 tests Pytest dans `tests_module.py`. Tous doivent retourner **PASSED**.

1. `test_weighted_avg_equal_weights — moyenne arithmétique simple`
2. `test_weighted_avg_double_weight_first`
3. `test_std_deviation_zero_when_constant`
4. `test_percentile_50_is_median`
5. `test_add_grade_rejects_value_above_20`

6. `test_student_average_single_course`
7. `test_student_general_uses_course_coeffs`
8. `test_class_ranking_sorted_desc`
9. `test_grade_distribution_buckets_sum_to_total`
10. `test_bulletin_includes_all_courses_with_grade`

10 Critères de validation et remise

Barème (sur 24)

Composante	Points
Module C — fonctions scalaires + batch, compile sans warning	4
Module Python — wrapper ctypes + logique métier	4
Tests Pytest — 10 tests PASSED	3
API Flask — tous les endpoints fonctionnels	4
Dashboard web — 3 graphiques opérationnels	3
GUI C# — communication API et écrans demandés	4
Benchmark — 3 scénarios + tableau + explication orale	2
Total	**/24**

Structure du dossier de remise

```
projet-8-notes_scolaires/
├── calculs.c
├── calculs.dll
├── module.py
├── tests_module.py
├── app.py
├── generator.py
├── benchmark.py
├── templates/
│   └── dashboard.html
├── data/                        (5e – JSON)   ou   db.sqlite   (6e – SQL)
└── GUI/
    └── Projet8.sln
```

Démonstration orale (5 minutes)

1. Lance ton API Flask et ton dashboard.
2. Lance le générateur de données et observe le dashboard se remplir.
3. Ouvre la GUI C# et démontre 2 cas d'usage.

4. Présente le résultat du benchmark et explique pourquoi le scénario batch est plus rapide.