PROJET D'EXAMEN · MULTI-LANGAGES

Projet 2
Suivi sportif & fitness

Journal d'entraînement avec IMC et calories

Domaine :	Santé · Sport · Suivi personnel
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 développes une application de suivi d'entraînement personnel. L'utilisateur enregistre ses séances sportives, l'application calcule les calories brûlées via le MET (Metabolic Equivalent of Task) et le poids, affiche l'IMC en continu et fournit des statistiques d'activité hebdomadaire et mensuelle.

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 :

- Gestion du profil utilisateur (poids, taille, âge, sexe).
- Ajout d'une séance (type d'activité, durée en minutes, date, intensité).
- Calcul automatique de l'IMC à chaque mise à jour du profil.
- Calcul des calories brûlées : kcal = MET × poids_kg × durée_h.
- Statistiques hebdomadaires et mensuelles (total min, kcal, nombre de séances).
- Catégorie IMC (insuffisance, normal, surpoids, obésité grade 1/2/3).
- Historique pondéral et tendance.

> **Tables MET (extraits) —** course: 8.0 · vélo: 6.0 · natation: 7.0 · musculation: 4.0 · yoga: 2.5 · marche rapide: 4.3

> **Catégories IMC (OMS) —** < 18.5 : insuffisance · 18.5-24.9 : normal · 25-29.9 : surpoids · ≥ 30 : obésité (3 grades)

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 calc_bmi(float weight_kg, float height_m);`	IMC = poids / taille²
`int bmi_category(float bmi);`	0=under · 1=normal · 2=over · 3=obese
`float calories_burned(float weight, float met, int minutes);`	kcal = MET × kg × h
`float met_for_activity(int type_id, float intensity);`	Table MET × facteur d'intensité
`void calc_bmi_batch(float* w, float* h, float* out, int n);`	IMC batch
`void calories_batch(float* w, float* met, int* mins, float* out, int n);`	kcal batch

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).

- `calc_bmi(weight_kg, height_m) -> float`
- `bmi_category_label(bmi: float) -> str`
- `calories_burned(weight, met, minutes) -> float`
- `save_user(data), get_user(id), list_users()`
- `add_session(user_id, data) -> dict`
- `list_sessions(user_id, week=None) -> list`
- `weekly_stats(user_id) -> dict — {total_min, total_kcal, count, by_type}`
- `bmi_history(user_id) -> list — [(date, bmi), ...]`

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/users`	Liste
POST	`/api/users`	Création
GET	`/api/users/{id}`	Détail (avec IMC courant)
PUT	`/api/users/{id}`	Mise à jour profil
GET	`/api/sessions?user_id=`	Liste filtrée
POST	`/api/sessions`	Ajout (calories calculées serveur)
GET	`/api/users/{id}/stats`	Statistiques hebdo/mensuelles
GET	`/api/users/{id}/bmi-history`	Historique IMC

6 Dashboard web — `/dashboard`

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

1. Minutes par jour de la semaine (bar chart).
2. Répartition par type d'activité (doughnut).
3. Évolution IMC dans le temps (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#.

- LoginForm — sélection utilisateur (ComboBox + nouveau).
- ProfileForm — édition profil + affichage IMC live + catégorie.
- AddSessionForm — type (ComboBox), durée (NumericUpDown), date.
- StatsForm — graphique simple (System.Windows.Forms.DataVisualization).
- HistoryForm — liste paginée des séances avec filtres.

8 Format des données

Variante 5e — Persistance JSON

```
{
  "user": {
    "id": 1, "name": "Marie", "weight_kg": 62, "height_m": 1.68,
    "age": 17, "sex": "F"
  },
  "session": {
    "id": 12, "user_id": 1, "type": "course",
    "duration_min": 45, "met": 8.0,
    "date": "2026-05-15", "calories": 372
  }
}
```

Variante 6e — Persistance SQL (MySQL/SQLite)

```
CREATE TABLE users (
  id          INTEGER PRIMARY KEY AUTOINCREMENT,
  name        TEXT NOT NULL,
  weight_kg   REAL NOT NULL,
  height_m    REAL NOT NULL,
  age         INTEGER,
  sex         TEXT CHECK (sex IN ('M','F'))
);
CREATE TABLE sessions (
  id           INTEGER PRIMARY KEY AUTOINCREMENT,
  user_id      INTEGER NOT NULL REFERENCES users(id),
  type         TEXT NOT NULL,
  duration_min INTEGER NOT NULL,
  met          REAL NOT NULL,
  date         TEXT NOT NULL,
  calories     REAL NOT NULL
);
```

9 Plan de tests Pytest

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

1. `test_bmi_basic` — 70 kg / 1.75 m → ≈ 22.86
2. `test_bmi_category_normal` — IMC 22 → category 'normal'
3. `test_bmi_category_underweight`
4. `test_bmi_category_obese_grade_2`
5. `test_calories_burned_running_30min`
6. `test_met_lookup_course`
7. `test_save_session_roundtrip`
8. `test_weekly_stats_aggregation`

9. `test_bmi_history_returns_chronological`
10. `test_validate_negative_weight_rejected`

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-2-fitness/
├── 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/
    └── Projet2.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.