PROJET D'EXAMEN · MULTI-LANGAGES

Projet 6
Caisse enregistreuse (POS)

Système Point of Sale pour petit commerce

Domaine :	Commerce · Vente · Comptabilité
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 réalises une caisse enregistreuse (Point of Sale) pour un petit commerce. Tu gères un catalogue de produits avec stock, tu encaisses des transactions multi-articles avec calcul de TVA par taux, et tu rends automatiquement la monnaie en pièces et billets EUR. Le dashboard donne le chiffre d'affaires quotidien et les produits les plus vendus.

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 produits : nom, prix HT, taux TVA (6 %, 12 %, 21 %), stock.
- Création d'une transaction (panier multi-produits).
- Calcul TVA séparé par taux (utile pour la déclaration TVA).
- Calcul du total TTC.
- Calcul du rendu de monnaie (en pièces et billets EUR).
- Décrément automatique du stock à chaque vente.
- Alerte si stock d'un produit < 5 unités.
- Statistiques quotidiennes : CA, nombre de tickets, top produits.

> **Taux TVA Belgique —** 6 % : alimentation, médicaments 12 % : restauration, charbon 21 % : taux normal (par défaut)

> **Dénominations EUR —** Billets : 500 · 200 · 100 · 50 · 20 · 10 · 5 Pièces : 2 · 1 · 0.50 · 0.20 · 0.10 · 0.05 · 0.02 · 0.01

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_vat(float price_ht, float vat_rate);`	Montant de TVA
`float calc_ttc(float price_ht, float vat_rate);`	Prix TTC
`int render_change(int cents, int* coins_out, int n_denom);`	Remplit tableau de quantités par dénomination
`float sum_total(float* prices, int n);`	Somme d'un panier
`void calc_ttc_batch(float* p, float* r, float* out, int n);`	Conversion HT → TTC 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_vat(price_ht, vat_rate) -> float — wrapper ctypes`
- `calc_ttc(price_ht, vat_rate) -> float — wrapper ctypes`
- `render_change(amount_cents) -> dict — {500: n, 200: n, …, 1: n}`
- `add_product(data), update_product(id, data), delete_product(id)`
- `list_products(), get_low_stock_alerts()`
- `create_transaction(items, amount_given) -> dict`
- `list_transactions(date=None) -> list`
- `daily_summary(date=None) -> dict — {ca_ttc, ca_ht, tva_total, n_tickets}`
- `top_products(date=None, n=10) -> list`

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/products`	Catalogue (filtre ?lowstock=true)
POST	`/api/products`	Création
PUT	`/api/products/{id}`	Modification
DELETE	`/api/products/{id}`	Suppression (refusée si stock > 0)
GET	`/api/transactions?date=`	Liste
POST	`/api/transactions`	Création + décrément stock + rendu
GET	`/api/stats/daily?date=`	Statistiques jour
GET	`/api/stats/top-products?n=10`	Top produits

6 Dashboard web — `/dashboard`

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

1. CA quotidien sur 30 jours (line chart).
2. Top 10 produits par chiffre d'affaires (bar chart horizontal).
3. Répartition CA par taux TVA (doughnut).

7 Interface C# — Windows Forms

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

- ProductsForm — CRUD catalogue (DataGridView).
- POSForm — caisse principale : sélection produit, panier live, total live.
- PaymentForm — saisie montant donné, affichage du rendu détaillé.
- ReceiptForm — affichage du ticket imprimable.
- StatsForm — CA jour/semaine/mois.

8 Format des données

Variante 5e — Persistance JSON

```
{
  "product": {
    "id": 1, "name": "Pain", "price_ht": 1.50,
    "vat_rate": 0.06, "stock": 120
  },
  "transaction": {
    "id": 42, "date": "2026-05-19T14:32:00",
    "items": [
      { "product_id": 1, "qty": 2, "unit_price_ttc": 1.59 },
      { "product_id": 7, "qty": 1, "unit_price_ttc": 4.84 }
    ],
    "total_ht": 7.20, "total_vat": 0.82, "total_ttc": 8.02,
    "amount_given": 10.00,
    "change": { "200": 0, "100": 1, "50": 1, "20": 2, "10": 0, "5": 0, "2": 4, "1": 0 }
  }
}
```

Variante 6e — Persistance SQL (MySQL/SQLite)

```
CREATE TABLE products (
  id        INTEGER PRIMARY KEY AUTOINCREMENT,
  name      TEXT NOT NULL,
  price_ht  REAL NOT NULL CHECK (price_ht >= 0),
  vat_rate  REAL NOT NULL CHECK (vat_rate IN (0.06, 0.12, 0.21)),
  stock     INTEGER NOT NULL DEFAULT 0 CHECK (stock >= 0)
);
CREATE TABLE transactions (
  id           INTEGER PRIMARY KEY AUTOINCREMENT,
  date         TEXT NOT NULL DEFAULT CURRENT_TIMESTAMP,
  total_ht     REAL NOT NULL,
  total_vat    REAL NOT NULL,
  total_ttc    REAL NOT NULL,
  amount_given REAL NOT NULL
);
CREATE TABLE transaction_items (
  id              INTEGER PRIMARY KEY AUTOINCREMENT,
  transaction_id  INTEGER NOT NULL REFERENCES transactions(id),
  product_id      INTEGER NOT NULL REFERENCES products(id),
  qty             INTEGER NOT NULL CHECK (qty > 0),
  unit_price_ttc  REAL NOT NULL
);
```

9 Plan de tests Pytest

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

1. `test_calc_vat_21_percent — 100 € × 21 % → 21.00 €`

2. `test_calc_ttc_basic — 100 € HT @ 21 % → 121.00 €`
3. `test_render_change_exact_amount — 0 piece à rendre`
4. `test_render_change_2_47 — 2 × 1 €, 4 × 10 c, 1 × 5 c, 2 × 1 c`
5. `test_render_change_greedy_minimum — minimum de pièces`
6. `test_create_transaction_decrements_stock`
7. `test_create_transaction_insufficient_stock_rejected`
8. `test_low_stock_alert_threshold_5`
9. `test_daily_summary_totals_match`
10. `test_top_products_sorted_by_revenue`

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-6-caisse_pos/
├── 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/
    └── Projet6.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.
