Énoncé du TP IGL

🎓 TP : The Refactoring Swarm

Durée	2 à 3 Semaines (Sprint Unique)
Organisation	Équipes de 4 étudiants
Évaluation	**100% Automatisée** (Performance & Data-Driven)

---

1. Contexte Scientifique 🔬

Vous participez à une expérience de recherche en Génie Logiciel Empirique.

L'objectif n'est pas seulement de produire du code, mais de concevoir une architecture d'agents autonomes (LLM) capable de réaliser de la maintenance logicielle sans intervention humaine.

Votre mission : Construire un système multi-agents, **"The Refactoring Swarm"**, capable de prendre en entrée un dossier contenant du code Python "mal fait" (buggé, non documenté, non testé) et de livrer en sortie une version propre, fonctionnelle et validée par des tests.

---

3. Architecture du système 🏗️

Votre système doit orchestrer la collaboration entre au moins 3 agents spécialisés :

1. **L'Agent Auditeur (The Auditor) :** Lit le code, lance l'analyse statique et produit un plan de refactoring.
2. **L'Agent Correcteur (The Fixer) :** Lit le plan, modifie le code fichier par fichier pour corriger les erreurs.
3. **L'Agent Testeur (The Judge) :** Exécute les tests unitaires.
   - *Si échec :* Il renvoie le code au Correcteur avec les logs d'erreur (Boucle de Self-Healing).
   - *Si succès :* Il valide la fin de mission.

2. Organisation de l'Équipe (4 Rôles) 👥

Ce projet nécessite des compétences variées. Répartissez-vous les rôles suivants dès le premier jour. Bien que chacun soit responsable de sa partie, le code final doit être intégré dans un seul dépôt Git.

1. **L'Orchestrateur (Lead Dev)**🧠
   - Conçoit le graphe d'exécution (via *LangGraph*, *CrewAI* ou *AutoGen*).
   - Gère la logique de passage de relais : *Quand passer de l'Auditeur au Correcteur ? Quand arrêter la boucle ?*
   - Responsable du main.py et de la gestion des arguments CLI.
2. **L'Ingénieur Outils (The Toolsmith)**🛠️
   - Développe les fonctions Python que les agents appellent (l'API interne).
   - Implémente la sécurité : interdiction pour les agents d'écrire hors du dossier "sandbox".
   - Gère les interfaces vers les outils d'analyse (pylint) et de test (pytest).
3. **L'Ingénieur Prompt (Prompt Engineer)**💬
   - Rédige et versionne les instructions systèmes (System Prompts).
   - Optimise les prompts pour minimiser les hallucinations et le coût en tokens.
   - Gère le contexte : s'assurer que l'agent a accès au code pertinent sans saturer sa mémoire.
4. **Le Responsable Qualité & Data (Data Officer)**📊
   - **CRITIQUE :** Responsable de la télémétrie. Il garantit que chaque action des agents est enregistrée dans le fichier logs/experiment_data.json selon le schéma imposé.
   - Crée le jeu de données de test interne pour valider que le système fonctionne avant la soumission.

Dans ce TP, votre code dépend du code des autres. Si l'Ingénieur Outils change le nom d'une fonction sans prévenir l'Ingénieur Prompt, l'agent cessera de fonctionner. Vous devez communiquer en permanence. Considérez que vous construisez une seule machine complexe, pas quatre petites machines séparées.

---

5. Critères d'évaluation automatisée 🤖

À la fin des 3 semaines, le Bot de Correction clonera votre dépôt et lancera votre système sur un **"Hidden Dataset"** (5 cas minimum de code buggé que vous n'avez jamais vus). La note est calculée automatiquement selon la formule suivante :

Dimension	Poids	Critères vérifiés par le Bot
Performance	**40%**	• Le code final passe-t-il les tests unitaires ? • Le score de qualité (Pylint) a-t-il augmenté ?
Robustesse Technique	**30%**	• Le système tourne-t-il sans planter ? • Pas de boucle infinie (max 10 itérations) ? • Respect de l'argument --target_dir.
Qualité des Données	**30%**	• Le fichier experiment_data.json est-il valide ? • Contient-il l'historique complet des actions ?

6. Démarche conseillé 📅

- **Etape 1 : Hello World & Outils**
  - Installation propre (Virtualenv). Lancez "python check_setup.py" pour valider.
  - Création des outils (lecture/écriture fichier, exécution pylint).
  - Premier agent simple qui analyse un fichier.
- **Etape 2 : La Boucle de Feedback**
  - Connexion : Auditeur -> Correcteur.
  - Gestion des itérations : Le correcteur doit pouvoir réessayer si l'auditeur n'est pas satisfait.
  - Optimisation des prompts par l'ingénieur Prompt.
- **Etape 3 : Tests & Robustesse**
  - Ajout de l'Agent Testeur (pytest).
  - Validation complète du format JSON des logs.
  - Test sur vos propres "fichiers pièges" pour vérifier que le système ne casse pas tout.

Politique Anti-Plagiat & Intégrité ⚠️

Ce projet fait l'objet d'une surveillance stricte pour garantir la validité des résultats de recherche.

1. **Code Similarity :** Tout le code Python sera analysé avec un outil de détection du plagiat
2. **Prompt Forensics :** Vos fichiers de prompts seront comparés. Une similarité forte (>70%) entre deux équipes est considérée comme du plagiat.
3. **Git History :** Nous analysons l'historique des commits. Un projet déposé en un seul commit le dernier jour sera rejeté (Note = 0).
4. **Log Signature :** Les logs contiennent des signatures uniques (ID/Timestamps). **Copier les logs d'un autre groupe est mathématiquement détectable et entraînera l'exclusion immédiate.**