Guide de Configuration Technique du TP IGL (2025-2026) : The Refactoring Swarm

Avant de coder, suivez ce guide scrupuleusement. Ce projet sert de base à une étude scientifique sur le génie logiciel assisté par IA.

> **Note sur la confidentialité :** Vos interactions avec le système (logs, fréquence des commits, prompts) seront récupérées et **anonymisées** avant analyse. Elles vont servir à comprendre les interactions techniques avec les LLM .

L'environnement est standardisé pour garantir que votre code tournera sur la machine du Correcteur Automatique. Pour que votre TP soit noté par le **Bot de Correction Automatique**, vous devez respecter cette structure à la lettre. Tout écart entraînera un échec de l'exécution (Note technique = 0).

1. Pré-requis Système

- **Python :** Version **3.10** ou **3.11** (⚠️ Python 3.12+ non supporté).
- **Git :** Installé.
- **Clé API :** Google Gemini.

2. Installation Pas à Pas

A. Cloner le Template (OBLIGATOIRE)

Ne partez pas de zéro. Un dépôt modèle (Template) est mis à votre disposition. Clonez-le pour démarrer.

- **git clone** https://github.com/sof-coder/refactoring-swarm-template.git
- **cd** refactoring-swarm-template

ci-dessous la structure du "Dépôt". Ne modifiez pas les noms des dossiers/fichiers marqués d'un cadenas (🔒).

```
/refactoring-swarm-template
│
├── main.py                🔒 Point d'entrée OBLIGATOIRE
├── requirements.txt       🔒 Liste des dépendances (ne pas supprimer les libs de base)
├── .env                   🔒 Vos clés API (à ne JAMAIS commiter)
├── check_setup.py         🔒 Script de vérification d'environnement
├── /src                   👈 Code source de vos agents et outils
├── /logs                  🔒 Dossier de sortie pour les données (géré par logger.py)
└── /sandbox               👈 Dossier de travail temporaire (là où le code est réparé)
```

B. Environnement Virtuel

Ne travaillez jamais sur votre Python global.

- **Windows :**
```
python -m venv venv
.\venv\Scripts\activate
```
- **Mac/Linux :**
```
python3 -m venv venv
source venv/bin/activate
```

C. Installer les dépendances

- **pip** install -r requirements.txt

D. Configuration des Clés API

1. Dupliquez le fichier `.env.example` et renommez-le `.env`
2. Vous pouvez obtenir une clé gratuite sur Google AI Studio.
3. Ajoutez votre clé (ne committez jamais ce fichier !) :
    - *(Gemini - Gratuit & Recommandé) :* `GOOGLE_API_KEY=AIzaSy...`

E. Hygiène Git (Crucial pour l'analyse)

Pour permettre l'analyse scientifique de votre progression :

- **Commits fréquents :** Faites un commit à chaque étape logique (ex: création d'un agent, correction d'un bug) plutôt qu'un seul gros commit à la fin.
- **Messages clairs :** Utilisez des messages descriptifs (ex: feat: ajout logic agent de test, fix: parsing json error).

3. Vérification ("Sanity Check")

Lancez le script de vérification inclus :

- python check_setup.py

Si vous voyez des ❌ rouges, ne commencez pas à coder. Corrigez d'abord l'environnement.

4. Protocole de Logging (Logging Protocol)

Ce projet sert de base à une étude scientifique. Pour que vos résultats soient analysables, vous devez respecter un protocole de logging strict. Le module src/utils/logger.py est fourni pour cela.

Règle d'Or : Chaque interaction significative avec le LLM (analyser, générer, corriger) doit être enregistrée.

A. Les Types d'Actions Standardisés

Vous ne devez pas inventer de noms d'actions. Utilisez impérativement la classe ActionType fournie pour catégoriser le travail de vos agents :

Action (Code)	Quand l'utiliser ?
ActionType.ANALYSIS	L'agent lit le code pour comprendre, vérifier les normes ou chercher des bugs. Aucune modification n'est faite.
ActionType.GENERATION	L'agent crée du contenu nouveau qui n'existait pas (ex: écrire des Tests Unitaires, générer de la doc).
ActionType.DEBUG	L'agent analyse une erreur d'exécution ou une stacktrace pour diagnostiquer un problème.
ActionType.FIX	L'agent réécrit une partie du code existant pour corriger un bug ou améliorer la qualité (Refactoring).

B. Contenu Obligatoire (Prompts)

Pour valider la qualité de votre "Prompt Engineering", le logger vérifiera automatiquement que vous fournissez bien :

1. **input_prompt** : Le texte exact envoyé au LLM.
2. **output_response** : La réponse brute reçue du LLM.

Si ces champs sont manquants, votre programme s'arrêtera avec une erreur.

C. Exemple d'implémentation

Voici comment appeler le logger dans vos agents :

Python

```python
from src.utils.logger import log_experiment, ActionType

# ... votre code d'agent ...

# Exemple d'appel OBLIGATOIRE après une interaction
log_experiment (
    agent_name = "Auditor_Agent",
    model_used = "gemini-2.5-flash",
    action = ActionType.ANALYSIS,  # <-- Utilisez l'Enum ici
    details = {
        "file_analyzed": "messy_code.py",
        "input_prompt": "Tu es un expert Python. Analyse ce code...", # OBLIGATOIRE
        "output_response": "J'ai détecté une absence de docstring...", # OBLIGATOIRE
        "issues_found": 3
    },
    status="SUCCESS" )
```

Le fichier `logs/experiment_data.json` sera généré automatiquement. Vérifiez régulièrement qu'il se remplit bien. Si ce fichier est vide ou mal formé à la fin, **la note "Qualité des Données" sera de 0**

5. Le Point d'Entrée (main.py)

Votre programme sera lancé par le **Bot de Correction Automatique** avec la commande suivante :

- `python main.py --target_dir "./sandbox/dataset_inconnu"`

Votre code doit lire cet argument, lancer les agents sur ce dossier, et s'arrêter proprement une fois fini.

6. Travailler en équipe sur le même code du TP 🤝

Travailler en équipe sur Git est différent de travailler seul. Si vous "commitez" tous sur le même fichier en même temps, vous allez créer des **conflits**. Suivez ce tutoriel pour gérer votre dépôt d'équipe. Vous pouvez utiliser la ligne de commande Git (voir section 6.1) ou l'environnement VS Code (voir section 6.2).

6.1. Tutoriel Git (Ligne de commande)

Étape A : Création du Dépôt d'Équipe

⚠️ **À faire par UNE SEULE personne (le "Chef d'équipe" Git) :**

1. Allez sur GitHub et créez un **Nouveau Repository** (vide). Nommez-le (ex: `Refactoring-Swarm-Equipe-01`).
2. Mettez-le en **Public**
3. Allez dans **Settings > Collaborators** et invitez les comptes GitHub de vos 3 camarades.

Sur votre PC (où vous avez déjà cloné le Template de l'enseignant), changez l'adresse distante pour pointer vers VOTRE nouveau dépôt :

Bash

```
# On supprime le lien vers le repo de l'enseignant
git remote remove origin
# On ajoute le lien vers votre repo d'équipe
git remote add origin https://github.com/USER_CHEF_EQUIPE/Refactoring-Swarm-Equipe-01.git
# On envoie tout le code template vers votre repo
git push -u origin main
```

Étape B : Rejoindre l'équipe (Les 3 autres membres)

Une fois invités, les 3 autres membres font simplement :

Bash

```
git clone https://github.com/USER_CHEF_EQUIPE/Refactoring-Swarm-Equipe-01.git
cd Refactoring-Swarm-Equipe-01
```

Étape C : La Routine Quotidienne (Le Workflow)

Ne travaillez jamais directement sur `main` si vous pouvez l'éviter. Utilisez des **Branches**.

1. Avant de commencer à coder (Le matin) : Toujours récupérer les dernières modifications des autres !

```bash
git checkout main
git pull origin main
```

2. Créer votre espace de travail : Créez une branche pour votre tâche (ex: `prompt-auditor`).

```bash
git checkout -b ma-feature-du-jour
```

3. Coder et Sauvegarder :

```bash
git add .
git commit -m "J'ai ajouté la fonction X et le prompt Y"
```

4. Partager votre travail :

```bash
# On envoie la branche sur GitHub
git push origin ma-feature-du-jour
```

Ensuite, allez sur GitHub et créez une "Pull Request" pour fusionner votre travail dans `main`.

Comment gérer les Conflits 🆘

Si vous ne voulez pas gérer de conflits de fusion compliqués **Chacun son fichier** (ex : L'Orchestrateur touche à `main.py`). Si vous devez modifier le fichier d'un autre : **Prévenez-le avant à travers votre outil de collaboration (ex : Discord/Slack)!**

6.2. Tutoriel Git avec VS Code : Travailler à 4 sans chaos

VS Code facilite la tâche grâce à son onglet ***"Contrôle de code source"*** (l'icône avec 3 points reliés). 

Étape A : Création du Dépôt d'Équipe (1er jour)

⚠️ À faire par UNE SEULE personne (le "Chef d'équipe") :

1. Allez sur le site **GitHub.com** et créez un **Nouveau Repository** (vide). Nommez-le (ex: `Refactoring-Swarm-Equipe-01`).
2. Copiez l'URL du dépôt (ex: `https://github.com/USER_CHEF_EQUIPE/Refactoring-Swarm-Equipe-01.git`).
3. Ouvrez votre projet local (le template de l'enseignant) dans VS Code.
4. Ouvrez l'onglet **Contrôle de code source** (Barre de gauche, icône branche) ou faites `Ctrl+Shift+G`.
5. Cliquez sur les **3 petits points (...)** en haut de ce panneau > **Remote** > **Add Remote...**
6. Collez l'URL de votre nouveau dépôt GitHub. VS Code demandera un nom : tapez `origin`.
7. Cliquez sur le bouton **Publier la branche** (le nuage en bas à gauche ou le bouton bleu).
    - *Si VS Code vous demande vos identifiants GitHub, connectez-vous.*
8. Enfin, sur le site GitHub, allez dans **Settings > Collaborators** et invitez vos 3 coéquipiers.

Étape B : Rejoindre l'équipe (Les 3 autres membres)

1. Attendez l'invitation par email et acceptez-la.
2. Ouvrez VS Code (fenêtre vide).
3. Sur la page d'accueil (ou via `F1` > *Git: Clone*), cliquez sur **Cloner le dépôt**.
4. Collez l'URL du dépôt de l'équipe et choisissez un dossier sur votre PC.

Étape C : La Routine Quotidienne (Le Workflow VS Code)

Ne codez jamais tous sur la branche `main` en même temps.

1. Avant de commencer (Le matin) : Regardez en bas à gauche de la fenêtre VS Code, à côté du nom de la branche (ex: `main`), il y a une icône de flèches circulaires (Synchroniser).

- Cliquez dessus pour **récupérer (Pull)** les dernières modifications des autres.

2. Créer votre espace de travail (Branche) :

- Cliquez sur le nom de la branche actuelle (en bas à gauche, écrit `main`).
- Dans le menu qui s'ouvre en haut, cliquez sur **+ Créer une nouvelle branche**.
- Donnez-lui un nom explicite (ex: `feature/prompts-auditeur`).

3. Sauvegarder votre travail (Commit) : Quand vous avez fini une tâche :

- Allez dans l'onglet **Contrôle de code source** (`Ctrl+Shift+G`).
- Vous verrez la liste des fichiers modifiés.
- Cliquez sur le **+** à côté des fichiers pour les "Stager" (les préparer).
- Écrivez un message clair dans la zone de texte (ex: *"Ajout du prompt système pour l'auditeur"*).
- Cliquez sur le bouton **Commit** (le V ✔️ ou le bouton bleu "Commit").

4. Partager votre travail (Push & Sync) :

- Cliquez sur le bouton bleu **Publier la branche** (ou **Synchroniser les modifications**).
- Votre code est maintenant sur GitHub !

5. Fusionner (Sur GitHub) :

- Allez sur GitHub, vous verrez un bouton vert "Compare & Pull Request". Cliquez dessus pour fusionner votre branche dans `main`.

Comment gérer les Conflits (Merge Conflicts) ? 🆘

Si VS Code vous dit "Conflit détecté" lors d'une synchronisation :

1. Les fichiers en conflit apparaissent en rouge avec un point d'exclamation (!) dans l'onglet Source Control.
2. Cliquez sur le fichier. VS Code affiche l'éditeur de conflit :
   - **Current Change (Vert) :** Ce que VOUS avez fait.
   - **Incoming Change (Bleu) :** Ce que VOTRE CAMARADE a fait.
3. Au-dessus du conflit, cliquez sur le petit texte cliquable :
   - *Accept Current Change* (Garder le vôtre).
   - *Accept Incoming Change* (Garder le sien).
   - *Accept Both Changes* (Garder les deux).
4. Une fois résolu, sauvegardez le fichier et faites un nouveau Commit.

Astuce Anti-Conflit : Si vous touchez au fichier `main.py`, demandez sur votre groupe de chat (ex : discord/slack) : *"Quelqu'un est sur le main.py en ce moment ?"*

7. Instructions de Rendu et Collecte de Données (⚠️ TRES IMPORTANT)

Ce TP nécessite la récupération de vos logs d'exécution pour valider votre travail et l'analyse scientifique.

1. **Vérification finale :** Avant de rendre, ouvrez `logs/experiment_data.json` et assurez-vous qu'il contient bien l'historique complet de vos tests avec les prompts.
2. **FORCER l'ajout des logs :** Par défaut, Git ignore souvent les fichiers de logs. Pour que votre travail soit validé, vous devez **forcer** l'envoi de ce fichier spécifique :

   ```
   Bash
   git add -f logs/experiment_data.json
   git commit -m "DATA: Submission of experiment logs"
   git push origin main
   ```

   (Ou via VS Code : Clic droit sur le fichier grisé `experiment_data.json` > "Stage Changes" (si disponible) ou via le terminal intégré).
3. **Soumission :** Déposez le lien de votre dépôt Git sur la plateforme de cours. *Si le fichier logs est absent de votre dépôt, la note "Qualité des Données" sera de 0.*