Technical Configuration Guide- IGL Lab (2025-2026) - The Refactoring Swarm

Before coding, follow this guide carefully. This project serves as the basis for a scientific study on AI-assisted software engineering.

> **Note on confidentiality:**Your interactions with the system (logs, commit frequency, prompts) will be retrieved and**anonymized**before analysis. They will be used to understand the technical interactions with LLMs.

The environment is standardized to ensure that your code will run on the Automatic Corrector machine.In order for your practical work to be graded by the**AutoCorrect Bot**You must adhere to this structure strictly. Any deviation will result in execution failure (Technical Score = 0).

1. System Prerequisites

- **Python :** Version **3.10**or**3.11**(⚠️ Python 3.12+ not supported).
- **Git :**Installed.
- **API key:** Google Gemini.

2. Step-by-Step Installation

A. Clone the Template (REQUIRED)

Don't start from scratch.A template repository is available. Clone it to get started.

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

below the structure du"Deposit". Do not change the names of folders/files marked with a padlock (🔒).

```
/refactoring-swarm-template
│
├── main.py                🔒 Mandatory entry point
├── requirements.txt       🔒 List of dependencies (do not delete the base libraries)
├── .env                   🔒 Your API keys (NEVER commit them)
├── check_setup.py         🔒 Environment check script
├── /src                   👈 Source code of your agents and tools
├── /logs                  🔒 Output folder for data (managed by logger.py)
└── /sandbox               👈 Temporary working folder (where the code is fixed)
```

B. Virtual Environment

Never work on your global Python environment.

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

C. Install the dependencies

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

D. API Key Configuration

1. Duplicate the file `.env.example` and rename it `.env`
2. You can get a free key on Google AI Studio.
3. Add your key (never commit this file!):
    - *(Gemini - Free & Recommended):* `GOOGLE_API_KEY=AIzaSy...`

E. Hygiene Git (Crucial for analysis)

To allow for the scientific analysis of your progress:

- **Frequent commitments:** Make a commit at each logical step (e.g., creating an agent, fixing a bug) rather than one large commit at the end.
- **Clear messages:** Use descriptive messages (e.g., feat: adding logic test agent, fix: parsing json error).

3. Verification ("Sanity Check")

Run the included verification script:

- python check_setup.py

If you see red ❌ symbols, don't start coding. Fix the environment first.

4. Protocole de Logging (Logging Protocol)

This project serves as the basis for a scientific study. For your results to be analyzable, you must adhere to a strict logging protocol. The module src/utils/logger.py is provided for this purpose.

Golden Rule: Each significant interaction with the LLM (analyze, generate, correct) must be recorded.

A. Standardized Action Types

You must not invent action names. You must use the class.ActionTypeprovided to categorize the work of your agents:

Action (Code)	When to use it?
ActionType.ANALYSIS	The agent reads the code to understand it, check for standards, or look for bugs. No modifications are made.
ActionType.GENERATION	The agent creates new content that did not exist (e.g., writing unit tests, generating documentation).
ActionType.DEBUG	The agent analyzes an execution error or a stack trace to diagnose a problem.
ActionType.FIX	The agent rewrites part of the existing code to fix a bug or improve quality (Refactoring).

B. Mandatory Content (Prompts)

To validate the quality of your "Prompt Engineering", the logger will automatically verify that you provide the following:

1. **input_prompt**The exact text sent to the LLM.
2. **output_response**: The raw response received from the LLM.

If these fields are missing, your program will stop with an error.

C. Example of implementation

Here's how to call the logger in your agents:

Python

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

# ... your agent code ...

# Example of a MANDATORY call after an interaction
log_experiment (
    agent_name = "Auditor_Agent",
    model_used = "gemini-2.5-flash",
    action = ActionType.ANALYSIS,  # <-- Use the Enum here
    details = {
        "file_analyzed": "messy_code.py",
        "input_prompt":"You're a Python expert. Analyze this code...", # MANDATORY
        "output_response":"I detected a missing docstring...", # MANDATORY
        "issues_found": 3
    },
    status="SUCCESS" )
```

The file`logs/experiment_data.json`will be generated automatically. Check regularly that it is being populated correctly. If this file is empty or malformed at the end,**The "Data Quality" score will be 0**

5. The Entry Point (main.py)

Your program will be launched by the**AutoCorrect Bot**with the following command:

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

Your code should read this argument, launch the agents on this case, and stop cleanly when finished.

6. Work as a team on the same code of theCity 🤝

Working as a team on Git is different from working alone. If you all commit to the same file at the same time, you will create...**conflicts**Follow this tutorial to manage your team repository. You can use the Git command line (see section 6.1) or the VS Code environment (see section 6.2).

6.1. Git Tutorial (Command Line)

Step A: Creating the Team Repository

⚠️ **To be done by ONE person only (the Git "Team Leader"):**

1. Go to GitHub and create a**New Repository**(empty). Name it (e.g.,`Refactoring-Swarm-Equipe-01`).
2. Put it in**Public**
3. Go to**Settings > Collaborators**and invite the GitHub accounts of your 3 classmates.

On your PC (where you have already cloned the teacher template), change the remote address to point to YOUR new repository:

Bash

```
# We remove the link to the teacher's repository
git remote remove origin
# We're adding the link to your team repo
git remote add origin https://github.com/USER_CHEF_EQUIPE/Refactoring-Swarm-Equipe-01.git
# We send all the template code to your repo
git push -u origin main
```

Step B: Join the team (The 3 other members)

Once invited, the other 3 members simply do the following:

Bash

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

Step C: The Daily Routine (The Workflow)

Never work directly on`main`If you can avoid it. Use**Branches**.

1. Before starting to code (In the morning): Always copy the latest changes from others!

```
Bash
git checkout main
git pull origin main
```

2. Create your workspace: Create a branch for your task (e.g.,`prompt-auditor`).

```
Bash
git checkout -b my-feature-of-the-day
```

3. Code and Save:

```
Bash
git add .
git commit -m "I added function X and prompt Y"
```

4. Share your work:

```
Bash
# We push the branch to GitHub
git push origin my-feature-of-the-day
```

Next, go to GitHub and create a "Pull Request" to merge your work into`main`.

How to Manage Conflicts 🆘

If you don't want to deal with complicated merge conflicts**Each to their own file**(e.g., The Orchestrator touches on`main.py`If you need to modify someone else's file:**Notify him beforehand through your collaboration tool (e.g., Discord/Slack)!**

6.2. Git Tutorial with VS Code: Working with 4 people without chaos

VS Code makes the task easier thanks to its tab***"Source code control"***(the icon with 3 connected dots). 

Step A: Creating the Team Depot (1is day)

⚠️ **To be done by ONE person only (the "Team Leader"):**

1. Go to the website**GitHub.com**and create a**New Repository**(empty). Name it (e.g.,`Refactoring-Swarm-Equipe-01`).
2. Copy the repository URL (ex: `https://github.com/USER_CHEF_EQUIPE/Refactoring-Swarm-Equipe-01.git`).
3. Open your local project (the teacher's template) in VS Code.
4. Open the tab**Source code control**(Left bar, branch icon) or do`Ctrl+Shift+G`.
5. Click on the**3 little dots (...)**at the top of this panel >**Remote** > **Add Remote...**
6. Paste the URL of your new GitHub repository. VS Code will ask for a name: type`origin`.
7. Click the button**Publish the branch**(the cloud in the bottom left corner or the blue button).
    - *If VS Code asks for your GitHub credentials, log in.*
8. Finally, on the GitHub website, go to**Settings > Collaborators**and invite your 3 teammates.

Step B: Join the team (The 3 other members)

1. Wait for the email invitation and accept it.
2. Open VS Code (empty window).
3. On the homepage (or via`F1` > *Git: Clone*), click on**Clone the repository**.
4. Paste the team's repository URL and choose a folder on your PC.

Step C: The Daily Routine (Workflow VS Code)

Never code everything on the branch`main` at the same time.

1. Before starting (In the morning): Look at the bottom left of the VS Code window, next to the branch name (e.g.:`main`), there is a circular arrows icon (Synchronize).

- Click on it to**retrieve (Pull)**the latest changes from others.

2. Create your workspace (Branch):

- Click on the name of the current branch (bottom left, written`main`).
- In the menu that opens at the top, click on**+ Create a new branch**.
- Give it a descriptive name (e.g.,`feature/prompts-listener`).

3. Save your work (Commit): When you have finished a task:

- Go to the tab**Source code control** (`Ctrl+Shift+G`).
- You will see the list of modified files.
- Click on the**+**next to the files for the "Stage" (to prepare them).
- Write a clear message in the text box (e.g.:*"Added prompt system for the listener"*).
- Click the button**Commit**(the V ✔️ or the blue "Commit" button).

4. Share your work (Push & Sync):

- Click on the blue button**Publish the branch**(or**Synchronize changes**).
- Your code is now on GitHub!

5. Merge (On GitHub):

- Go to GitHub, you'll see a green "Compare & Pull Request" button. Click it to merge your branch into`main`.

How to manage Merge Conflicts? 🆘

If VS Code tells you "Conflict detected" during a synchronization:

1. Conflicting files appear in red with an exclamation mark (!) in the Source Control tab.
2. Click on the file. VS Code displays the conflict editor:
    - **Current Change (Vert) :**What YOU did.
    - **Incoming Change (Bleu) :**What YOUR COMRADE did.
3. Above the conflict, click on the small clickable text:
    - *Accept Current Change*(Keep yours).
    - *Accept Incoming Change*(Keep your own).
    - *Accept Both Changes*(Keep both).
4. Once resolved, save the file and make a new commit.

Anti-Conflict Tip: If you touch the file`main.py`Ask on your chat group (e.g., Discord/Slack):*"Is anyone working on main.py right now?"*

7. Rendering Instructions and Data Collection (⚠️ VERY IMPORTANT)

This lab requires you to retrieve your execution logs to validate your work and the scientific analysis.

1. **Final check:**Before returning, open`logs/experiment_data.json`and make sure that it contains the complete history of your tests with the prompts.
2. **FORCE the addition of logs:**By default, Git often ignores log files. For your work to be committed, you must**force**sending this specific file:

    Bash

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

    (Or via VS Code: Right-click on the greyed-out file)`experiment_data.json`> "Stage Changes" (if available) or via the integrated terminal).
3. **Submission :**Upload the link to your Git repository to the course platform.*If the logs file is missing from your repository, the "Data Quality" score will be 0.*