Here is a detailed, step-by-step guide for fine-tuning your project framework, including prerequisites for both Windows and Mac environments.

Pre-requisites

1. Install and Setup Python

- **Windows:**
  - Download the latest Python installer from python.org.
  - Run the installer and select “Add Python to PATH”.
  - Verify installation via Command Prompt:

    ```
    python --version
    pip --version
    ```

- **Mac:**
  - Use Homebrew (recommended) by running:

    ```
    /bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"
    brew install python
    ```

  - Alternatively, download the installer from python.org.
  - Verify installation in Terminal:

    ```
    python3 --version
    pip3 --version
    ```

2. Import the Maven Selenium Framework Project

- Clone or import your Maven-based Selenium framework into your preferred IDE (e.g., IntelliJ, Eclipse, VS Code).
- Ensure that the project builds successfully and that dependencies (e.g., Selenium WebDriver, TestNG) are correctly configured.

Steps to Fine-Tune the Project Framework

Step 1: Understand the Framework

- **Study the Project Structure:**
  - Review framework methods, existing pages, and overall architecture.
  - Understand key classes (e.g., ProjectSpecificMethods, Reporter) and patterns (e.g., Page Object Model).
- **Identify Unique Fields:**
  - List the unique fields and elements that are common across pages (minimum three per type).

Step 2: Collect Data

- **Analyze the Page Codebase:**
  - Inspect each page in your framework and identify key DOM elements.
- **Capture DOM for Every Element:**
  - For each unique element, capture its HTML snippet.
- **Format Data:**
  - Place the captured HTML into a file using markers:
    - Use `===HTML===` before the DOM snippet.
    - Use `===JAVA===` before the corresponding Java code snippet.
  - Ensure you have at least 10 examples covering all types of elements (e.g., text fields, buttons, dropdowns).
- **Organize Examples:**
  - Save each example in a `.txt` file within a dedicated `training` folder.

Step 3: Convert Data into JSONL Format

- **Run the Conversion Script:**
  - Use a Python script (e.g., generate_json.py) that reads each .txt file from the training folder and converts them to chat-formatted JSONL entries.
  - Verify that the output file, for example, is named training_data_chat.jsonl.
- **Check Format:**
  - Ensure each line in the JSONL file contains a valid JSON object with a "messages" array (system, user, assistant).

Step 4: Provide the JSONL File to Babu

- **File Handoff:**
  - Send the training_data_chat.jsonl file to Babu.
- **Model Creation:**
  - Note that the API for upload is not open yet – Babu will manually upload the file and create a fine-tuned model.
  - Babu will then update the model name (e.g., “TL DB”) for your access.
  - Record the reported training loss for future reference.

Step 5: Update and Run the Browser Extension

- **Update Extension:**
  - Replace your current browser extension with the latest version provided by Babu.
  - Under the provider configuration (Testleaf / ft:gpt), update the model name with the fine-tuned model's name.
- **Configure API Key:**
  - Log in to api.testleaf to get your API key and update it in the extension configuration.
- **Validation:**
  - Capture screens of the existing pages (for baseline verification).
  - Run the extension to generate new pages and download them.
  - Validate the correctness of generated pages.
  - Add the validated pages to your Selenium framework and build tests using these pages.
  - Record any issues that arise; these will help you decide if retraining is needed.

Step 6: Calculate the Cost

- **Dashboard Review:**
  - Log into the api.testleaf dashboard and review the cost incurred for testing your fine-tuned model.
- **Security:**
  - Once done, delete the API key if it was exposed to others to maintain security.
