1. Project Vision & Core Concept

The objective is to build a **Web-based Testing Agent** that acts as a capable partner for QA Engineers. This project is not about building a generic script; it is about creating a "Human-in-the-Loop" assistant that solves real-world testing challenges: exploration, test design, and implementation.

The exact implementation details—how data flows, how state is managed, and how the UI behaves—are open to innovation, provided the system fulfills the functional goals described below in their chronological sequence.

System Architecture Goals:

- **Interface:** A web-based chat interface facilitating a dialogue between the Human Tester and the Agent.
- **Visual Context:** The Agent must control a visible browser instance. The user should see the work happening in real-time to build trust.
- **Observability & Metrics:** The system must provide insight into the "Agent's Brain."
  - Must display granular metrics: **Average Response Time (per iteration/page)** and **Tokens Consumed (per iteration/page)**.
  - Visualization using tools like MLFlow, LangFuse, or custom dashboards.

2. Functional Workflow (Chronological & Dependent)

The project follows a linear user journey. Each phase produces an output that serves as the necessary foundation for the next. **The format of these outputs is flexible, as long as they effectively enable the subsequent phase.**

Phase 1: Exploration & Knowledge Acquisition

- **Goal:** The Agent must ingest a URL and obtain a deep understanding of the page's structure, logic, and interactivity.
- **Creative Opportunity:** How does the Agent "see"? Does it rely on the DOM? Screenshots? A hybrid approach? Does it click around randomly or follow heuristics?
- **Required Outcome:** The Agent must produce a structured representation of the page.
  - This output must be detailed enough to serve as the "ground truth" for generating test cases later.
  - It should capture element candidates (locators) and potential descriptions.

Phase 2: Collaborative Test Design

- **Goal:** To reach an agreement between the Human and the AI on *what* needs to be tested to ensure high coverage.
- **The Problem to Solve:** How do we visualize coverage?
  - The Agent should propose a list of logical test cases (e.g., a table or list).
- **Interaction:** The user acts as a reviewer. The Agent refines the plan based on feedback.

Phase 3: Implementation (Code Generation)

- **Goal:** Generate executable, clean, and maintainable test code e.g. (Playwright + Python).
- **Creative Opportunity:**
  - **Locator Strategy:** How does the Agent select the best locator? (ID vs. CSS vs. XPath vs. Semantic).
  - **Self-Correction:** How does the Agent verify the code works *while* writing it?

Phase 4: Verification & Trust Building

- **Goal:** Prove to the user that the tests pass and verify the expected behavior.
- **The Output:** The user needs evidence.
  - This could be a report with screenshots, a generated video of the execution, or a step-by-step log of actions.
  - The "Review" phase is critical—the user must be able to critique the run, and the Agent must be able to refactor based on that critique.

3. Additional Creative Challenges & System Utilities

- **Context Management:** Capabilities to "Reset" the Agent to a clean slate.
- **Model Constraints:**
  - **No Paid Providers:** Teams are restricted from using paid-subscription-only LLM providers for the final delivery. The solution must utilize Free Tier APIs (e.g., Gemini Free, Hugging Face), Open Source models, or Local LLMs.

4. Evaluation Criteria

General Compliance: The evaluation will assess the successful implementation of **all** functional requirements, architectural constraints, and deliverables outlined in the document above (Sections 1, 2, and 3).

The specific grading pillars are as follows:

1. **Accuracy & Truthfulness:**
   - **Hallucinations:** accuracy for generating locators, steps, or elements that do not exist on the page and will depend on the approach.
   - **Test Validity:** Generated tests must accurately verify the intended behavior and will depend on the approach.
2. **Student Competence & Code Defense:**
   - **Code Understanding:** Students should understand their code during discussion.
   - **Model Compliance:** Verification that no paid-subscription LLM APIs were used for the final delivery.
3. **Feature Completeness & Usability:**
   - **Workflow Completion:** Successful execution of all phases (Explore → Design → Implementation → Verification).
   - **UX:** The system must provide a transparent, intuitive interface for the user.
4. **Metrics & Observability:**

- **Visibility:** The system must clearly display **Average Response Time** and **Token Usage** (per iteration or page).
- **Performance:** Evaluation of the execution speed for "Explore" phases.