

AI Agent to Test Websites Automatically Using Natural Language

1. Introduction / Objective

This project implements an intelligent agent capable of performing automated end-to-end (E2E) testing on web applications. The agent accepts natural language instructions, interprets them, generates Playwright test scripts, executes those scripts in a headless browser, and produces detailed test reports.

2. Methodology / Workflow

Instruction Interpretation Phase

- Accepts user-defined natural language test cases.
- Parses input and identifies actionable test steps.

Code Generation Phase

- Converts interpreted steps into executable Playwright scripts.
- Dynamically generates assertions to validate expected outcomes.

Execution Phase

- Runs Playwright tests in a headless browser environment.
- Monitors DOM and adapts dynamically to web structure changes.

Reporting Phase

- Summarizes results including passed/failed assertions.
- Generates human-readable test reports for review.

3. Modules

Instruction Parser Module: Interprets natural language test descriptions and maps them to browser actions.

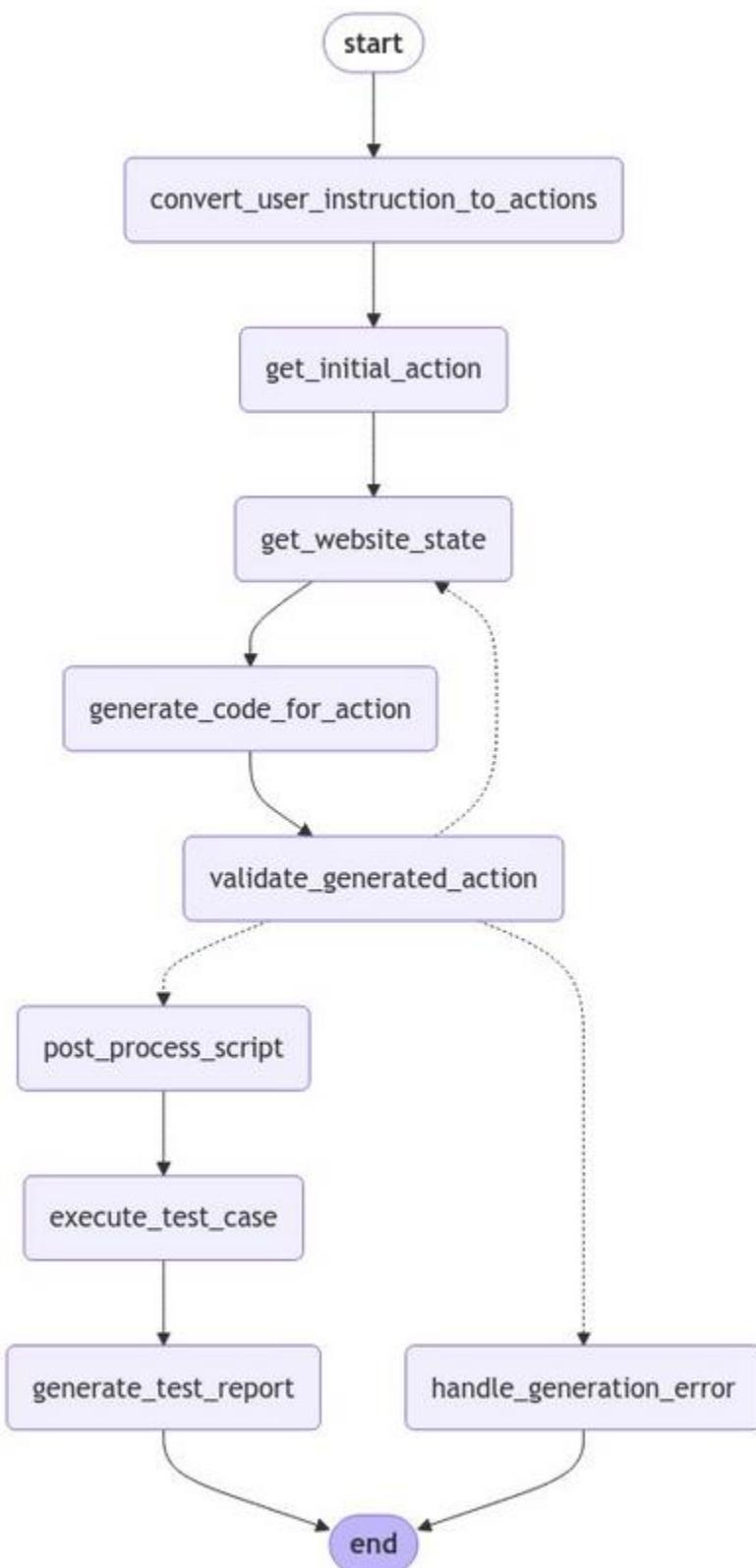
Code Generation Module: Converts parsed actions into executable Playwright Python scripts.

Execution Module: Runs tests headlessly and captures runtime logs.

Assertion & Reporting Module: Validates expected results and compiles test summaries.

4. System Design / Architecture

The architecture integrates LangGraph for agent workflow orchestration and Playwright for browser automation. Each test cycle follows a pipeline of instruction → code generation → execution → reporting.



5. Week-wise Module Implementation and High-Level Requirements

Milestone 1: Week 1–2

- Set up Python environment and install dependencies (LangGraph, Playwright, Flask).
- Define project structure and initialize Flask server for static HTML test page.
- Implement baseline LangGraph agent configuration for handling user inputs.

Milestone 2: Week 3–4

- Build the Instruction Parser Module for interpreting natural language test cases.
- Map parsed actions into structured commands.
- Establish LangGraph workflow connecting parser → code generator.
- Validate basic test case conversion output.

Milestone 3: Week 5–6

- Develop Playwright Code Generation Module to automate script creation.
- Implement assertion generator for result validation.
- Integrate Playwright execution logic for headless browser runs.
- Test sample cases on local HTML forms to verify functional accuracy.

Milestone 4: Week 7–8

- Add Reporting Module to capture and format test results.
- Implement advanced error handling and adaptive DOM mapping.
- Finalize UI Implementation: Design and implement the polished **front-end** to display the final structured test report, including success/failure statuses and execution details.
- Finalize end-to-end execution workflow from input to test to report.
- Prepare final documentation and testing demonstration.

6. Technology Stack

Programming Language: Python 3.x

Frameworks / Tools:

- LangGraph
- Playwright (Python version)
- Flask
- LangChain Community Toolkit (PlaywrightBrowserToolkit)
- Streamlit/Gradio