AI Exploratory Testing Agent

Project Guide & Specification

Complete documentation for an AI-assisted browser testing agent that explores authorised web apps, finds evidence-backed bugs, and produces reproducible reports.

Version 0.1 · Companion to the downloadable project zip

Requirements · Architecture · Safety · API · Commands · Acceptance criteria

Contents

1. Project overview and purpose
2. Technology stack
3. Functional & non-functional requirements
4. Safety, privacy and test boundaries
5. Architecture
6. Agent loop
7. Database schema
8. Project folder structure
9. Backend implementation notes
10. Frontend / dashboard notes
11. API contract (summary)
12. Bug report output format
13. Commands to run the project
14. Test strategy & acceptance criteria

1. Project overview and purpose

The AI Exploratory Testing Agent explores authorised web applications, discovers evidence-backed defects, and produces reproducible reports. It combines deterministic checks with an optional local LLM (Ollama) and a Playwright-driven browser.

Goals

- Explore only targets the user has explicitly authorised.
- Stay within a strict action budget and host allowlist.
- Capture screenshots and structured evidence for every finding.
- Produce JSON and HTML reports that can be re-run or handed to a human.
- Never execute arbitrary code or unrestricted URLs from page content.

2. Technology stack

Layer	Choice	Notes
Backend API	FastAPI + Uvicorn	Async, OpenAPI docs at /docs
Browser	Playwright (Chromium)	Headless by default
Storage	SQLite	runs, actions, findings, evidence
Optional LLM	Ollama (local)	Structured suggestions only
Frontend	HTML / CSS / JS	Static dashboard, no build step
Language	Python 3.10+	Backend + scripts

3. Functional & non-functional requirements

Functional

- Start a run against a user-supplied URL only after explicit authorisation.
- Enforce max actions, per-action timeout, and overall run timeout.
- Observe page state (links, forms, console errors, network failures).
- Select next actions via rules; optionally ask a local LLM for suggestions.
- Validate every action against the safety policy before execution.
- Run deterministic checks after each action and capture evidence.
- Persist runs, actions and findings in SQLite; export JSON/HTML reports.
- Allow the user to stop a run; reliably close browser contexts.

Non-functional

- Works offline once dependencies are installed (except optional Ollama).
- Clear, accessible UI with loading/empty/error states.
- Sensitive values (passwords, tokens, cookies) redacted from logs/reports.
- No execution of arbitrary code or shell commands from page content.

4. Safety, privacy and test boundaries

- Require explicit authorisation checkbox before any run starts.
- Restrict navigation to an allowlist of hosts (default: localhost / 127.0.0.1).
- Block javascript:, data:, file:, mailto: and similar schemes.
- Enforce action budget and per-action timeout.
- Provide cancellation and close browser contexts on stop.
- Never let the LLM execute arbitrary code, shell commands or unrestricted URLs.
- Treat webpage text as untrusted; ignore embedded instructions that try to change agent policy.
- Redact passwords, tokens, cookies and sensitive values from logs/reports.

5. Architecture

User → Dashboard (HTML/CSS/JS) → FastAPI Backend → Run Manager + Safety Policy → Exploration Agent (+ optional Ollama) → Playwright Browser → Deterministic Checks + Evidence → SQLite → HTML/JSON Report Generator → Dashboard.

The agent loop is: Validate URL & authorisation → Observe page → Choose action (rules/LLM) → Safety-policy validation → Perform action → Deterministic checks → Capture evidence if needed → Persist → Budget remaining? → Repeat or Generate report.

6. Agent loop (simplified)

```
START
Validate URL + authorisation + allowlist
Observe page (DOM, links, forms, console, network)
Is there an allowed unvisited action?
  NO  → Finish exploration → Generate report → END
  YES → Select action (rules / optional LLM)
        Safety-policy validation
        Allowed?
          NO  → Reject + log → back to Observe
          YES → Perform action (wait / timeout)
                Run deterministic checks
                Finding suspected?
                  YES → Capture evidence → Persist finding
                  NO  → Record action outcome
                Budget remaining?
                  YES → Observe next state
                  NO  → Generate report → END
```

7. Database schema (SQLite)

runs: id, url, status, actions_taken, findings_count, started_at, finished_at, error

findings: id, run_id, title, category, severity, confidence, description, target_url, evidence, verification

8. Project folder structure

```
ai-exploratory-testing-agent/
■■■ backend/
```

```
■   ■■■ app/
■   ■   ■■■ main.py              # FastAPI entry
■   ■   ■■■ config.py            # Settings & limits
■   ■   ■■■ api/routes.py        # HTTP endpoints
■   ■   ■■■ agent/               # Loop + planner
■   ■   ■■■ browser/             # Playwright session
■   ■   ■■■ checks/              # Deterministic checks
■   ■   ■■■ safety/              # Allowlist / policy
■   ■   ■■■ models/              # Pydantic schemas
■   ■   ■■■ services/            # Run manager
■   ■   ■■■ db/                  # SQLite setup
■   ■■■ tests/
■   ■■■ requirements.txt
■■■ frontend/                   # Dashboard
■   ■■■ index.html
■   ■■■ css/style.css
■   ■■■ js/app.js
■■■ sample_app/                 # Safe local demo target
■■■ .env.example
■■■ .gitignore
■■■ README.md
■■■ COMMANDS.md
■■■ start.sh
```

9. Backend implementation notes

- FastAPI app with CORS for the static dashboard origin.
- RunManager coordinates BrowserSession, SafetyPolicy and the agent loop.
- SafetyPolicy blocks off-allowlist hosts and dangerous URL schemes.
- Deterministic checks cover console errors and failed actions; extend as needed.
- Evidence is stored as screenshot paths + metadata; keep retention policy explicit.

10. Frontend / dashboard notes

- Static HTML/CSS/JS – no build step required.
- New-run form: URL, authorisation checkbox, max actions, headless, optional LLM.
- Live status polling; Stop button cancels the active run.
- Findings table and JSON report download.

11. API contract (summary)

Method	Path	Purpose
POST	/api/runs	Start a run (requires authorised=true)
GET	/api/runs/{id}	Run status
POST	/api/runs/{id}/stop	Cancel run
GET	/api/runs/{id}/findings	List findings
GET	/api/runs/{id}/report	JSON/HTML report
GET	/health	Health check

Interactive docs: http://127.0.0.1:8000/docs after starting the server.

12. Bug report output format

```
Finding ID: BUG-0001
Title: [Specific defect title]
Target URL: [URL]
Category: Navigation / Console / Network / Form / UI / Other
Severity: Low / Medium / High / Critical
Confidence: 0.00–1.00
Verification: Confirmed / Suspected / Needs manual review
Environment: Browser, OS, timestamp, run ID
Preconditions: [Required setup]
Steps to reproduce:
 1. [Action]
 2. [Action]
Expected result: [What should happen]
Actual result: [What happened]
Evidence: screenshot path, console/network notes
Notes: [Optional]
```

13. Commands to run the project

First-time setup (macOS / Linux)

```
unzip ai-exploratory-testing-agent.zip
cd ai-exploratory-testing-agent

python3 -m venv .venv
source .venv/bin/activate
pip install --upgrade pip
pip install -r backend/requirements.txt
python -m playwright install chromium

cp .env.example .env
```

Start the backend API

```
cd backend
uvicorn app.main:app --reload --host 127.0.0.1 --port 8000
```

Serve the frontend dashboard (new terminal)

```
cd frontend
python3 -m http.server 5500
# open http://127.0.0.1:5500
```

Optional – sample app as safe target (new terminal)

```
cd sample_app
python3 -m http.server 3000
# use http://127.0.0.1:3000 as target URL in the dashboard
```

Quick start after first setup

```
./start.sh
# or:
source .venv/bin/activate
cd backend && uvicorn app.main:app --reload --host 127.0.0.1 --port 8000
```

14. Test strategy & acceptance criteria

- Unit tests: URL validation, allowlist, action policy, severity rules, report formatting, DB repositories.
- Integration tests: API endpoints, SQLite persistence, cancellation, report download.
- Browser tests: only against the bundled local demo app; verify safe navigation and screenshots.
- Safety tests: reject unsupported schemes, unapproved hosts, out-of-budget actions.
- UI tests: validation, progress, stop action, finding detail and exports.
- Acceptance: a run against sample_app produces at least one deterministic finding (e.g. console error or broken link) with a screenshot and a downloadable JSON report.

This guide accompanies the downloadable project zip. Place the PDF at downloads/AI_Exploratory_Testing_Agent_Project_Guide.pdf so the documentation page download button works.