Captain Fix- The ultimate quality project

Fix.AI Accelerator by **BRIGHTWAY** is an intensive, 120-hour modular program that equips participants with cutting-edge skills in software testing, automation, and artificial intelligence. The accelerator blends theoretical training with extensive hands-on practice, guiding participants through QA fundamentals, Python-based automation, and AI-driven test development before culminating in a high-stakes final project.

At the heart of the accelerator is the Autonomous QA Captain Challenge. Participants, divided into three competing teams, are tasked with building an AI-based autonomous "captain" that:

1. Accepts a website as input.
2. Generates a structured test plan.
3. Executes automated tests using frameworks like Selenium.
4. Logs defects into project management tools (e.g., Jira, Trello).
5. Exports test plans and results into Excel.
6. Sends a consolidated email report to the end user.

The competition ensures that participants not only learn core QA and automation skills but also apply them to real-world, end-to-end systems where AI orchestrates planning, execution, and reporting. Industry judges evaluate the projects based on technical robustness, coverage, execution quality, and innovation, crowning the team that builds the "smartest captain."

The **Fix.AI Accelerator** is designed for students, graduates, and early-career engineers who aspire to combine QA expertise with automation and AI capabilities. By the program's end, graduates will have both portfolio-ready project experience and the confidence to take on roles in QA automation, AI-driven testing, and related software engineering fields

Development Steps — Autonomous QA Captain

1. Input & Setup

- Build a CLI or simple UI where the user provides:
    - Website URL to test.
    - Optional parameters (depth, number of test cases, email address, Jira/Trello choice).
- Validate input (e.g., ensure site is accessible).

2. Test Plan Generation (Planner)

- Crawl or sample the website (basic sitemap/top links).
- Use an LLM (structured prompt) to generate a TestPlan JSON schema with:
    - Suites (Smoke, Navigation, Forms).
    - Test cases: IDs, steps, expected outcomes, priorities.
- Save the plan as both JSON and Excel (Plan.xlsx).

3. Test Execution (Executor)

- Parse the TestPlan JSON.
- Use Playwright or Selenium + pytest to run the cases:
    - Navigate, click, fill forms, and assert content.
- Collect outputs: pass/fail, error messages, screenshots, logs.
- Store results in structured Results.json and Results.xlsx.

4. Evidence Collection

- For each test case:
    - Screenshot(s).
    - Console or network logs.
    - Execution duration.
- Organize in /evidence/<caseId>/….

5. Reporting (Reporter)

- Generate:
    - Plan.xlsx – with cases and steps.
    - Results.xlsx – with outcomes, errors, and evidence links.
    - Summary.md / HTML – overview of run.
- Package outputs (zip if needed).

6. Integrations (Notifiers)

- Jira/Trello:
    - Create one epic/task per failed case.
    - Include repro steps + attach evidence.
- Email (SMTP):
    - Subject: “Captain Results — <site> — <timestamp>”.
    - Body: summary table + links.
    - Attach Excel files and evidence (if small enough).

7. End-to-End Flow

End-user runs one command:
run_captain --target=https://example.com --email=user@domain.com --pm=jira

- System auto-generates plan → executes tests → reports results → updates Jira/Trello → emails the user.

8. Team Development Milestones

1. Scaffold project + CLI.
2. Implement TestPlan schema + LLM planner.
3. Build an executor to test cases reliably.
4. Add a reporter for Excel + summary.
5. Connect integrations (Jira/Trello, email).
6. Polish UX (logging, configs, error handling).
7. Final demo preparation: run on a known site and show the full flow.

Tech Stack for Autonomous QA Captain

Core Language & Frameworks

1. Python 3.11+
    - Primary development language (covered in Module 2).
    - Strong ecosystem for automation (pytest, Selenium, Playwright) and AI (LangChain, OpenAI, DeepSeek).
2. LangChain
    - Framework for integrating LLMs.
    - Used in the Planner module to structure test plans (JSON output).
    - Enables prompt engineering, tool calling, and chaining logic.

Automation & Testing

- Selenium
    - For browser automation, locating elements, and executing test plans.
- pytest
    - For structured test execution and reporting.
- pydantic
    - For validating and managing structured data schemas (TestPlan, RunResult).

Data & Storage

- Excel (pandas + openpyxl)
    - For generating Test Plan.xlsx and Results.xlsx.
- MySQL (optional)
    - Only if persistence beyond files is needed (e.g., storing historical runs, evidence paths, test coverage stats).
    - Lightweight DB alternative: SQLite (no setup required).

Integrations

- Jira / Trello REST APIs
    - For task/bug creation with evidence attached.
- SMTP (Python smtplib) / SendGrid API
    - For emailing results to the user.

AI & Knowledge

- LLMs (OpenAI GPT, DeepSeek, Claude)
    - For generating plans, summarizing results, and structuring bug reports.
- Vector Store (optional) - Use Pinecone if an advanced RAG is needed later (not MVP-critical).

DevOps & Environment (optional)

- GitHub + CI (GitHub Actions) (optional)
    - For collaborative work across teams.
- Docker (optional)
    - To package the environment (Python + Selenium drivers + dependencies).