Solution Brief: Autonomous QA Agent (AQA)

The Problem: Test Generation is a Bottleneck

Manually generating and executing comprehensive tests is often the most time-consuming step in the development cycle. It demands significant developer focus, can introduce human error, and slows down Continuous Integration/Continuous Deployment (CI/CD) pipelines.

The Solution: AI Autonomous QA Agent (AQA)

The AQA is a low-code workflow designed to **autonomously generate, execute, and report** on tests for any code repository, transforming a manual, slow process into a fast, automated service.

Component	Function	Technology Stack
Workflow Engine	Orchestrates file fetching, LLM interaction, code execution, and reporting.	**n8n** (Low-Code Automation)
Intelligence	Analyzes code, generates test cases (descriptions), and produces executable test code.	**LLM API** (e.g., Gemini, GPT)
Execution	Clones the repo, runs the generated test files, and captures the output logs.	**n8n Execute Command** / Containerized Environment

---

The AQA Workflow: 4 Steps to Automated QA

The agent operates as a seamless, linear pipeline triggered by a simple input (a GitHub URL).

1. **Input & Ingestion:**
    - **Action:** User provides a **GitHub Repository URL** or a file path.
    - **n8n** uses a **Git Node** or shell command to clone and fetch the repository contents.
2. **LLM Analysis & Generation:**
    - **Action:** The system feeds relevant source code files (e.g., a function or module) to the **LLM API**.
    - **Output:** The LLM returns a structured response containing:
        - A list of **High-Level Test Cases** (descriptions).
        - The corresponding **Executable Test Code** (e.g., Python `pytest` or JS `jest` files).
3. **Test Execution:**
    - **Action: n8n** writes the generated test code into the cloned directory.
    - An **Execute Command** node is triggered to run the appropriate test runner (e.g., `pytest`, `jest`) against the newly created test file.
4. **Reporting & Delivery:**
    - **Action:** The execution logs are captured and analyzed.
    - **Output:** A clean, actionable report is compiled and delivered via a preferred channel (e.g., Slack, Email, or a database entry), detailing **Pass/Fail status** and error logs.

Key Advantages & Future Enhancements

Core Advantages Today

- **Speed:** Reduces test generation and execution from hours to minutes.
- **Coverage:** Ensures higher test coverage by leveraging AI's ability to analyze complex logic and edge cases.
- **Developer Focus:** Frees up developer time from writing boilerplate tests for more complex feature development.

Suggested Enhancements

- **Multi-Language Support:** Expand the LLM prompting to better handle and distinguish between multiple programming languages (e.g., Python, Node.js, Go).
- **Dependency Management:** Automatically detect required dependencies (`requirements.txt`, `package.json`) and install them before execution using the n8n workflow.
- **Self-Correction Loop:** If a test execution fails due to a syntax error in the generated test code, feed the error log *back* into the LLM and ask it to **debug and correct the test code** autonomously before re-executing.

Next Steps

This one-pager outlines the core value proposition. The next steps are:

1. **Proof of Concept (POC):** Build a minimum viable workflow to validate the core 4-step process.
2. **API Integration:** Secure and configure a robust LLM API endpoint for the workflow.
3. **Pilot Program:** Test the AQA on a small, contained internal repository.

Questions: Which internal repository offers the best starting point for a successful POC?