AI QA Engineers Assignment

Welcome to the AI QA Engineering Challenge.
This assignment focuses on evaluating your **technical skills** and ability to build, orchestrate, and validate **AI-driven systems** across modern LLM workflows, backend tooling, and browser-based automation. You will design tools, control LLM behavior through prompts, integrate everything using the **Model Context Protocol (MCP)**, and validate the system end-to-end using **Playwright UI automation**.

You will be assessed on your ability to work across:

- **LLM-driven tool invocation** (tool selection, argument generation, structured outputs)
- **MCP flows** (client ↔ server communication, multi-server tooling, schemas)
- **Backend & Python engineering**
- **API and external data integration**
- **UI automation and validation using Playwright**

The assignment is divided into **three progressive tasks**, each reflecting a core responsibility of an AI QA engineer:

1. **Task 1: Tool Creation & LLM-Driven Tool Invocation**
   You will design and implement AI tools (summarization, API fetching, evaluation) and an LLM-driven flow that interprets natural language prompts, selects tools, generates arguments, and produces structured outputs — without MCP involvement.

2. **Task 2: Build a Minimal MCP Ecosystem**
   You will expose the tools created in Task 1 through multiple MCP servers and implement an MCP client capable of discovering tools, invoking them manually or via LLM-generated calls, and validating structured responses.

3. **Task 3: Playwright UI Automation**
   You will build Playwright automation that interacts with real public web pages, extracts content, sends it to your Summarization MCP server/tool, and performs schema and correctness assertions on the returned output.

This challenge aims to bring out your best in **design, implementation quality, and testing**.

Good luck!

Table of Contents

Task 1. Tool Creation & LLM-Driven Tool Invocation

Create the required tools and implement an LLM-based flow that generates tool calls and sets tool arguments using prompts.

This task focuses on tool design, prompt clarity, structured outputs, and correct tool invocation, without MCP involvement.

You should create the following tools:

A. Summarization Tools

Given a document (PDF or raw text), the server must:

- Extract text.
- Chunk the content.
- Summarize each chunk.
- Produce a combined summary.

Must expose these tools:

- extract_pdf_text
- summarize_text
- summarize_pdf
- detect_language

A.1 Details per tool

Tool: extract_pdf_text

- Accepts a local path and HTTP URL. ?
- Returns extracted text and number of pages.
- Handles unreadable PDFs gracefully.
- Detects empty PDFs or PDFs with no textual content.

Tool: summarize_text

- Summarizes the text using LLMs (Large Language Models).
- Use system prompts to guide the summarization process.
  - Summarizes input text using LLMs.
  - Uses a system prompt to guide summarization.
- Returns structured JSON including:
  - Summary.
  - Prompt (system + user prompts sent to the model).
  - Metadata.

Tool: summarize_pdf

- Internally calls ***extract_pdf_text*.**
- Splits text into chunks.
- Summarizes each chunk using ***summarize_text***.
- Merges summaries.
- **Bonus**: Streams results:
  - {"chunk": 1, "partial_summary": "..."}
  - {"final_summary": "..."}

Tool: detect_language()

Simple language detection is acceptable.

B. API Fetching Tools

These tools interact with public APIs to fetch live data. Each tool returns structured JSON output, including the fetched information and metadata about the API call.

API	Description
Weather API	Fetches current weather for a specified city using a public weather API. *Returns temperature, wind speed, and weather conditions.*
Exchange Rate API	Fetches the latest exchange rate for a specified currency pair using a public API. *Returns the exchange rate and timestamp.*

Exposed Tools

Tool	Description
fetch_weather	Retrieves the current weather for a given city. Accepts city
fetch_exchange_rate	Retrieves the latest exchange rate for a given currency pair.

Prompts should be **clear and specific**, specifying:

- The exact data to return (e.g., temperature, wind speed, exchange rate)
- Any constraints, such as Units (e.g., Celsius, Fahrenheit), and Formatting rules for output.
- Timestamp.

Output

Each tool must return **structured JSON** containing:

1. Fetched Information.
2. API Call details.

C. Evaluation Tools

Tool: Evaluate_llm_responses

- It accepts ground truth and response and calculates:
    - Cosine Similarity.
    - Lexical Similarity.
    - Answer Conciseness.

Tool: hallucination_checker

- Using LLM: It accepts ground truth and response and calculates if there is any info mentioned in the response that is not included in the ground truth.

LLM Tool Invocation

- Implement an LLM layer that:
    - Receives a natural language prompt
    - Determines which tool(s) to use
    - Generates valid tool arguments from the prompt
    - Executes the selected tool(s)
- Prompts should be **clear and specific**, describing:
    - What data to return
    - Any constraints (units, limits, etc.)

Output

Define clear JSON schemas for:

- Tool outputs
- Prompt details (when applicable)
- Execution metadata

Task 2. Build a Minimal MCP Ecosystem

Expose the tools created in Task 1 through the MCP servers and implement an MCP client to interact with them.

This task focuses on MCP integration, multi-server communication, and schema validation.

A. MCP Client

Requirements

- Connects to multiple MCP servers.
- Support listing available tools.
- Call the MCP tools manually or use LLM to construct the tool call.
- Provides a clean CLI structure.
- Provide enough visibility during execution.
- **Bonus**: Support streaming.

B. MCP Summarization Server

Must expose these tools from Task1:

- extract_pdf_text
- summarize_text
- summarize_pdf
- detect_language

C. MCP External API Tooling Server

Must expose these tools from Task1:

- fetch_weather
- fetch_exchange_rate

D. MCP Evaluation Tool

Must expose these tools from Task1:

- evaluate_llm_responses
- hallucination_checker

Task 3. Playwright UI Automation

You should write Playwright tests to interact with public web pages, extract text, and send it to the summarization MCP tool for testing your server.

Use: https://en.wikipedia.org/wiki/Artificial_intelligence

Deliverables:

- Infrastructure for UI automation and content retrieval.
- Playwright browser usage (use Firefox).
- Locators for elements (search input, search result, content section, etc.).
- Structured assertions.

Required flow (Baseline Test)

1. Navigate to the Wikipedia home.
2. Locate and interact with the search input.
3. Enter a term and submit.
4. Click the first search result.
5. Scroll to a specific section heading.
6. Extract paragraph text beneath that heading.
7. Send the extracted text to your Summarization MCP server.
8. Perform valid assertions for output correctness & structure.

Additional Testing Expectations

Beyond the required flow, you are encouraged to expand the test suite to cover additional scenarios, such as:

- Different search terms or pages.
- Missing or empty extracted content.
- Large text inputs.
- Invalid or unexpected MCP responses.
- Schema validation failures.
- Partial or streaming responses (if supported).
- Error handling and retry behavior.

You are **not** required to implement all of these, but your choices should demonstrate sound QA judgment and prioritization.

Assignment Instructions

You should consider:

- Clear test structure.
- Usage of utilities.
- Use SOLID design principles.
- Assertions, logging, and failure reasoning.

Anyone should be able to:

1. Install
2. Run
3. Test
4. Understand

with minimal effort and **without asking questions**.

Deliverables

You should provide the following items as part of this assignment:
1. Source Code

1. Fully working implementation of all MCP components.
2. Playwright automation tests.
3. Clear folder structure.
4. Reusable utilities.

---

2. Requirements File

A complete list of all Python dependencies:

- requirements.txt (Python servers)

---

3. Usage / Setup Documentation

A simple usage guide describing:

- How to install dependencies.
- How to start the MCP servers.
- How to run the client.
- How to run Playwright tests.
- How to send a sample request.
- Example inputs and expected outputs.

A single Markdown file is enough:

- USAGE.md or README.md

---

4. Working Examples

Provide example commands and sample responses:

- Example tool invocations.
- Example API calls.
- Example summarization output.
- Example evaluation output.
- Example Playwright test output.

---

5. Packaged / Executable Form (Bonus)

- MCP servers are packaged (pip installable or containerized)
- Package dependencies into virtual environments.
- Client can be installed and run as a CLI tool (python -m mcp)
- Playwright tests run using a single command (pytest -q)

---

6. Screenshots or Demo (Bonus)

- Screenshots of the UI flow tested.
- Short screen recording.
- Example streaming responses.
