QA AI Automation System: Requirements Document

Version: 1.0 **Date:** September 3, 2025

1. Introduction

1.1. Project Overview

This document outlines the functional and non-functional requirements for the "QA AI Automation System." The system is designed to revolutionize the software testing lifecycle by allowing QA engineers and developers to write test cases in natural language. An AI-powered engine will interpret these test cases, automatically generate executable test scripts, run them against the target application, and provide detailed reports. The goal is to accelerate the testing process, reduce the technical barrier to creating automated tests, and improve overall software quality.

1.2. Scope

The scope of this project includes:

- A web-based user interface for managing projects and test cases.
- An AI core responsible for Natural Language Processing (NLP) to understand test steps.
- A code generation module to convert processed language into test scripts (e.g., Selenium, Cypress, Playwright).
- A test execution engine to run individual test scripts and entire test suites.
- A reporting dashboard to visualize test results, logs, and analytics.

Out of Scope:

- Mobile application testing (initial version).
- Performance, load, or security testing (initial version).
- Direct integration with every possible CI/CD tool (a generic API will be provided).

1.3. Intended Audience

This document is intended for project stakeholders, including project managers, software developers, QA engineers, and UI/UX designers.

2. Functional Requirements

2.1. User & Project Management

- **FR-001: User Authentication:** Users must be able to create an account and log in securely.
- **FR-002: Project Creation:** Authenticated users must be able to create projects to house their test suites and test cases. Each project will be associated with a specific application under test (AUT).
- **FR-003: Project Configuration:** For each project, users must be able to configure essential details, such as the application's base URL and the target testing framework (e.g., Selenium with Python, Cypress with JavaScript).

2.2. Test Case Creation from Natural Language

- **FR-010: Natural Language Input:** The system must provide a user-friendly text editor where users can write test steps in plain English. The system should handle both structured, multi-step formats and more casual, single-sentence instructions.
    - *Example 1 (Structured):*

```None
Test Case: Successful User Login
1. Navigate to the login page.
2. Enter "testuser" into the username field.
3. Enter "password123" into the password field.
4. Click the "Login" button.
5. Verify that the user is redirected to the dashboard.
6. Check that the text "Welcome, testuser" is visible.
```

-
    Example 2 (Casual):

```None
Go to the login page, enter the credentials for "testuser", and
ensure you are able to log in successfully.
```

-
  FR-011: AI-Powered Parsing: The AI engine must parse the natural language input to identify actions (e.g., "navigate", "enter", "click"), elements (e.g., "username field", "Login button"), and assertions (e.g., "verify", "check that", "ensure").
- **FR-012: Element Identification:** The system should provide a mechanism for users to map abstract element names ("username field") to concrete selectors (e.g., CSS

selectors, XPaths). This could be an interactive element picker or a simple key-value store.

2.3. Test Script Generation

- **FR-020: Automated Code Generation:** Based on the parsed natural language and element mappings, the system must automatically generate a clean, readable, and executable test script in the language and framework specified in the project configuration.
- **FR-021: Code Preview:** Users must be able to view the generated test script before execution.
- **FR-022: Code Editing (Optional but Recommended):** Users should have the option to manually edit the generated script to handle complex logic or edge cases that the AI might not cover.

2.4. Test Execution

- **FR-030: Single Test Case Execution:** Users must be able to trigger the execution of a single test case on demand.
- **FR-031: Real-time Logging:** The system must display real-time logs during test execution, showing which step is currently running and its status (pass/fail).
- **FR-032: Headless & Headed Execution:** The system should support both headless (no browser UI) and headed (with browser UI) execution modes for debugging purposes.

2.5. Test Suite Management

- **FR-040: Test Suite Creation:** Users must be able to group multiple test cases into a "Test Suite."
- **FR-041: Test Suite Execution:** Users must be able to execute an entire test suite with a single click.
- **FR-042: Execution Scheduling:** Users should be able to schedule test suite executions to run at specific times or on a recurring basis (e.g., nightly).

2.6. Reporting and Analytics

- **FR-050: Execution History:** The system must maintain a history of all test runs, including timestamps, duration, and overall status (pass/fail).
- **FR-051: Detailed Test Reports:** For each run, a detailed report must be generated, including:
  - A step-by-step breakdown of results for each test case.
  - Screenshots captured at the point of failure.
  - Error messages and stack traces for failed steps.
- **FR-052: Analytics Dashboard:** A dashboard must provide high-level insights, such as pass/fail rates over time, most frequently failing tests, and average execution times.

3. Non-Functional Requirements

3.1. Performance

- **NFR-001: Script Generation Speed:** The AI engine should generate a test script from a typical (10-15 step) test case in under 10 seconds.
- **NFR-002: UI Responsiveness:** The web interface must be responsive and load all pages within 3 seconds under normal network conditions.

3.2. Usability

- **NFR-010: Intuitive Interface:** The UI must be clean, modern, and intuitive, requiring minimal training for a new QA engineer to get started.
- **NFR-011: Clear Feedback:** The system must provide clear and immediate feedback for user actions, such as saving a test case or starting an execution.

3.3. Reliability

- **NFR-020: System Uptime:** The system should have an uptime of 99.5%.
- **NFR-021: Error Handling:** The system must handle unexpected errors gracefully without crashing, providing informative error messages to the user.

3.4. Security

- **NFR-030: Data Protection:** All user data, especially credentials for the application under test, must be encrypted both in transit and at rest.
- **NFR-031: Access Control:** Users should only be able to access projects and test cases they have created or have been granted permission to view.

3.5. Compatibility

- **NFR-040: Browser Support:** The web interface must be compatible with the latest versions of major web browsers (Chrome, Firefox, Safari, Edge).
- **NFR-041: Framework Support:** The initial version of the system must support generating test scripts for at least two of the following: Selenium (Python/Java), Cypress, or Playwright.

4. System Architecture & Technology Stack (Proposed)

- **Frontend:** A modern JavaScript framework like React or Angular for a dynamic user experience.
- **Backend:** A robust framework like Node.js (Express), Python (Django/Flask), or Go.
- **AI/NLP Engine:**
  - Utilize a powerful Large Language Model (LLM) via API (e.g., Gemini, GPT-4) for its superior natural language understanding capabilities.

    - Fine-tune a smaller, open-source model for more specific, domain-adapted tasks if needed.
- **Database:** A combination of a relational database (e.g., PostgreSQL) for structured data (users, projects) and a NoSQL database (e.g., MongoDB) for semi-structured data (test results, logs).
- **Test Execution Environment:** A containerized environment (e.g., using Docker) to run tests in isolated and consistent settings.
- **Message Queue:** A system like RabbitMQ or Kafka to manage test execution jobs asynchronously.

5. User Roles

- **Tester:** Can create and manage test cases, execute tests, and view reports within their own projects.
- **Admin:** Has full access to all projects and user accounts. Can manage system-level settings and configurations.