JuanTM

Test Automation Strategy Plan

Release 01

Context

As a seasoned software Test Automation S, I have often been tasked with evaluating the merits of Selenium-based versus non-Selenium frameworks. To avoid this fight, **I propose** a forward-thinking strategy that harnesses the synergies of both technologies, utilizing CLI as the linchpin to create a comprehensive and versatile test automation framework. Furthermore, I have identified the need to automate backend tests, which requires the unique strengths of both Cypress and Selenium. Cypress's exceptional ability to handle API and backend testing, combined with Selenium's robustness in handling complex browser interactions, makes a compelling case for a hybrid approach. By integrating both technologies, I aim to create a test automation framework that can efficiently and effectively automate a wide range of backend tests, leveraging the best of both worlds.

Scope

The scope of this project is to architect a test automation framework that leverages the strengths of Selenium and Cypress, adhering to best practices for a minimum viable product. The framework will be designed to automate regression tests exclusively for PostgreSQL-based SQL scenarios, with a focus on creating modular, reusable, consistent, and independent tests that can be easily integrated into continuous integration and continuous deployment (CI/CD) pipelines.

Test Automation Framework Structure

The framework structure will be created using the official Selenium CLI, and will consist of the following components:

- Node.js backend project: The framework will be built using Node.js, which will provide a scalable and efficient environment for test automation.
- PostgreSQL database: The framework will use a PostgreSQL database to store test data and results.
- Selenium: Selenium will be used for UI testing, providing a comprehensive testing approach for web applications.
- Cypress: Cypress will be used for API testing, providing fast and efficient testing for API endpoints.

Test Automation Framework

-

Test Automation Approach

-

Test Data Management

Test Data Overview

- The purpose of test data management is to ensure that the testing process has access to relevant, accurate, and complete data to support testing activities.
- Test data will be used to validate the functionality, performance, and security of the application.

Test Data Requirements

- The following types of data are required for testing:
    - Customer information (e.g., names, addresses, contact details)
    - Order data (e.g., order numbers, product details, quantities)
    - Product information (e.g., product names, descriptions, prices)
    - User authentication data (e.g., usernames, passwords)
- The volume of data required for testing is estimated to be [insert volume, e.g., 1000 customer records, 500 order records].
- The complexity of data required for testing includes [insert complexity, e.g., relationships between customer and order data, product variations].

Test Data Sources

- Test data will be generated using Faker and Mocks libraries.
- Additional data sources may include [insert additional sources, e.g., production data, third-party APIs].

Test Data Storage and Retrieval

- Test data will be stored in [insert storage mechanism, e.g., database, files, APIs].
- Test data will be retrieved and used during testing through [insert retrieval mechanism, e.g., automated scripts, manual data entry].

Test Data Maintenance

- Test data will be updated and refreshed regularly to ensure it remains relevant and accurate.
- Test data will be backed up and version-controlled to ensure data integrity and consistency.

Test Data Security

- Test data will be protected and secured to prevent unauthorized access and data breaches.

- Test data will be anonymized and masked to protect sensitive information.

Test Data Metrics and Reporting

- Test data metrics will be collected and reported to measure test data quality and effectiveness.
- Test data reports will be generated to provide insights into test data usage and trends.

Test Environment Management

Test Automation Metrics and Reporting

Test Automation Maintenance and Updates

Conclusion

Selenium and Cypress are powerful tools for automating web browsers and ensuring the quality of web applications. By leveraging these tools, teams can create robust and reliable tests that simulate real-user interactions, reducing the risk of defects and improving overall application performance. With Selenium's flexibility and Cypress's ease of use, teams can efficiently test complex web applications and deliver high-quality software products.

Selenium and Cypress are essential tools for any web development team, providing a robust and efficient way to test web applications. By integrating these tools into their testing strategy, teams can ensure that their applications meet the highest standards of quality and reliability. With Selenium's flexibility and Cypress's ease of use, teams can create tests that are both comprehensive and maintainable.

Test Automation Strategy Overview

The goal is to create a framework that is modular, reusable, consistent, independent, and has logging and reporting features.

Key Characteristics of the Automation Framework

The framework should have the following characteristics:

- **Modular**: Adaptable to change, allowing testers to modify scripts as per environment or login information changes.
- **Reusable**: Commonly used methods or utilities should be written in a common file, accessible to all scripts.
- **Consistent**: The suite should be written in a consistent format, following accepted coding practices.
- **Independent**: Scripts should be written to be independent of each other, ensuring that one test failure does not hold back the remaining test cases.
- **Logger**: Logging feature should be implemented to detect errors and their locations.
- **Reporting**: Reporting feature should be automatically embedded into the framework, sending results and reports via email.
- Integration: The framework should be easy to integrate with other applications, such as continuous integration or triggering automated scripts after deployment.

Benefits of Combining Selenium and Cypress

Some benefits of combining Selenium and Cypress include:

- Compatibility: Both Selenium and Cypress are widely used and compatible with various browsers and environments.
- Comprehensive testing: Combining Selenium and Cypress allows for a more comprehensive testing approach, covering both UI and API testing.
- Faster test execution: Cypress is known for its fast test execution, which can be beneficial for large test suites.
- Easy integration: Both Selenium and Cypress have large communities and are easy to integrate with other tools and frameworks.

Test Deliverables

- Test Cases
  - List of test cases with unique identifiers
  - Description of each test case, including:
    - Preconditions
    - Steps to reproduce
    - Expected results

Test 1: Create Table with Valid Columns

- Test Case: Create a table with valid columns (e.g., `id`, `name`, `email`)
- Preconditions:
  1. Admin user is logged in
  2. Database connection is established
- Steps:
  1. Send a `CREATE TABLE` query with valid columns
  2. Verify the table is created successfully
  3. Verify the columns are created with the correct data types
- Cypress Code:

Test 2: Create Table with Invalid Columns

- Test Case: Create a table with invalid columns (e.g., duplicate column names)
- Preconditions:
  1. Admin user is logged in
  2. Database connection is established
- Steps:
  1. Send a `CREATE TABLE` query with invalid columns
  2. Verify the table creation fails with an error
- Cypress Code:

Test 3: Create Table with Missing Columns

- Test Case: Create a table with missing columns (e.g., no primary key)

- Preconditions:
    1. Admin user is logged in
    2. Database connection is established
- Steps:
    1. Send a `CREATE TABLE` query with missing columns
    2. Verify the table creation fails with an error
- Cypress Code:

These tests cover basic scenarios for creating tables with valid and invalid columns, as well as missing columns. You can add more test cases to cover additional scenarios.

Note: You'll need to modify the `cy.exec` commands to use your actual PostgreSQL database connection and query execution methods.

Would you like me to elaborate on any of these tests or provide additional examples?

Implementation Plan

The implementation plan will consist of the following steps:

1. Week 1-2: Set up the Node.js backend project with a PostgreSQL database and integrate Selenium and Cypress using the official Selenium.
2. Week 3-4: Create the User Management Test Suite and implement logging and reporting features.
3. Week 5-6: Create the Product Management Test Suite and implement logging and reporting features.
4. Week 7-8: Create the Order Management Test Suite and implement logging and reporting features.
5. Week 9-10: Create the Payment Gateway Test Suite and implement logging and reporting features.
6. Week 11-12: Integrate the framework with continuous integration and continuous deployment (CI/CD) pipelines to automate test execution and deployment.

Deliverables

The deliverables for this project will include:

- A fully functional test automation framework that combines Selenium and Cypress technologies using a Node.js backend project with a PostgreSQL database.
- A comprehensive regression test suite that covers different endpoints, including user management, product management, order management, and payment gateway.
- A detailed report on the implementation plan, including the scope, framework structure, test suites, and deliverables.