Test Automation Solution Document

Project Overview

The goal of this project is to develop a test automation framework for UI testing using Java, Selenium, Cucumber, and TestNG. The framework will allow for the execution of automated tests for web applications and provide comprehensive reporting capabilities.

Assumptions

- The web application under test is a calculator application with basic arithmetic operations.
- The target browser for testing is Google Chrome.
- The test environment includes Java Development Kit (JDK), Maven, and Chrome browser.
- Automated WebDriver management is done by using WebDriverManager package.
- Test scenarios are written in Gherkin syntax using Cucumber.
- TestNG is used for test execution.
- The framework supports parallel test execution with a thread count of 4.
- The project is integrated with CircleCI for continuous integration.
- Git is used for version control.
- GitHub is used for source code management and collaboration.

Out of Scope Functionalities

- **Calculator History**

  The framework does not cover the functionality related to the calculator history. Testing of the calculator history feature is considered out of scope for this project.

- **Advanced Arithmetic Operations**

  Only basic arithmetic operations (addition, subtraction, multiplication, division) are covered in the test scenarios. Advanced arithmetic operations such as exponentiation, square root, etc., are not included in the scope of this project.

- **Expression and Result Validation**

  The final expression validation and result validation logic is currently mentioned in the feature file. However, to enhance scalability, it is recommended to move this logic away from the feature file and implement it within the framework. Due to time constraints, this enhancement is considered out of scope for the current phase of the project.

Steps Taken

1. **Project Setup:**
   - Created a Maven project structure with appropriate directories for source code, resources, and test files.
   - Configured Maven dependencies for Selenium WebDriver, Cucumber, and TestNG in the pom.xml file.
   - Initialized a Git repository for version control.

2. **Environment Setup:**
    - Installed JDK and Maven on the local development machine.
    - Downloaded Google Chrome web browser.

3. **Test Design:**
    - Identified and documented test scenarios based on the requirements.
    - Created feature files in Gherkin syntax to describe the behaviour of the calculator application.

4. **Implementation:**
    - Developed step definitions in Java to define the actions for each step in the feature files.
    - Implemented hooks to manage setup and teardown activities, including browser initialization and screenshot capture on test failure.
    - Created a Selenium Wrapper class to encapsulate WebDriver interactions and simplify test script development.
    - The framework uses PicoContainer for dependency injection to manage depenedencies such as WebDriver instance.
    - Developed utility classes for common functionalities such as WebDriver initialization and config file reading.

5. **Source Code Management:**
    - Initialized a Git repository for version control.
    - Committed and pushed all the project files to a remote repository in GitHub.

6. **Integration with CircleCI:**
    - Added CircleCI configuration file (.circleci/config.yml) to the project for continuous integration.
    - Configured CircleCI pipeline to build and test the project on every commit.

7. **Test Execution:**
    - Executed tests locally using Maven commands from the command line.
    - Ran tests from the IDE using TestNG runner classes and TestNG XML configuration files.
    - Additionally, created a .bat file for running the tests.
    - Verified test execution and reporting on CircleCI.

8. **Reporting:**
    - Generated HTML reports using Cucumber to provide detailed test results and execution status (results/test-result.html).
    - Included screenshots in the reports to facilitate troubleshooting and analysis.

    Sample Test Result Passed:

    Sample_Passed_T
    Result.html

Sample Test Result Failed:

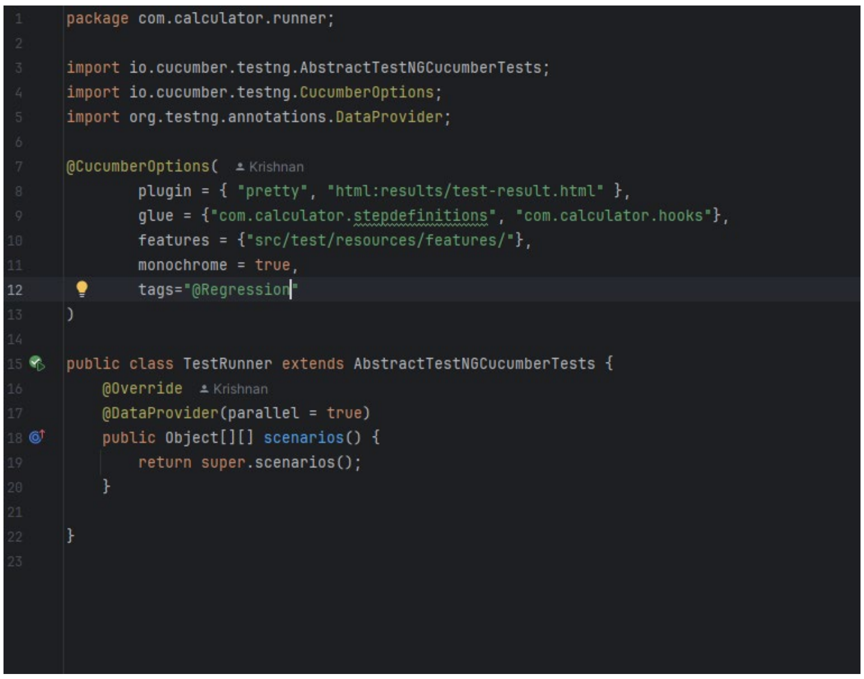
Sample_Failed_Te
Result.html

Code/Scripts Developed

1. **TestRunner.java:** TestNG runner class to execute Cucumber tests.
2. **CalculatorTest.java:** Step definitions class to define test steps.
3. **Calculator.feature:** Feature file for defining test scenarios in Gherkin format.
4. **CalculatorPage.java:** Class encapsulates the locators to interact with calculator buttons.
5. **DriverFactory.java:** Utility class for WebDriver management.
6. **CalculatorUtil.java:** Utility methods for converting operators to calculator symbols.
7. **SeleniumWrapper.java:** Wrapper class for Selenium WebDriver to simplify interactions.
8. **TestHooks.java**: Hook class for easy management of setup and teardown activities.

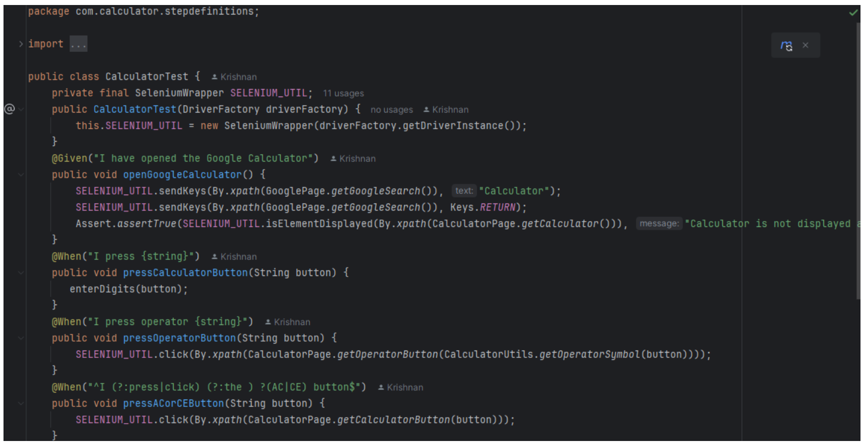
Code Snippets:

TestRunner.java



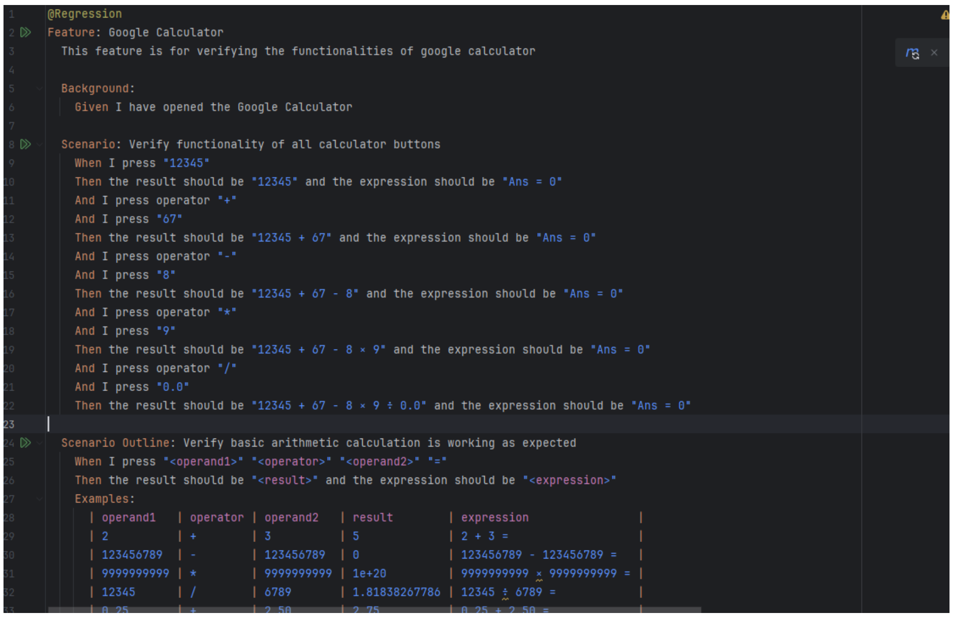
```java
package com.calculator.runner;

import io.cucumber.testng.AbstractTestNGCucumberTests;
import io.cucumber.testng.CucumberOptions;
import org.testng.annotations.DataProvider;

@CucumberOptions(  Krishnan
        plugin = { "pretty", "html:results/test-result.html" },
        glue = {"com.calculator.stepdefinitions", "com.calculator.hooks"},
        features = {"src/test/resources/features/"},
        monochrome = true,
        tags="@Regression"
)

public class TestRunner extends AbstractTestNGCucumberTests {
    @Override  Krishnan
    @DataProvider(parallel = true)
    public Object[][] scenarios() {
        return super.scenarios();
    }

}
```

CalculatorTest.java



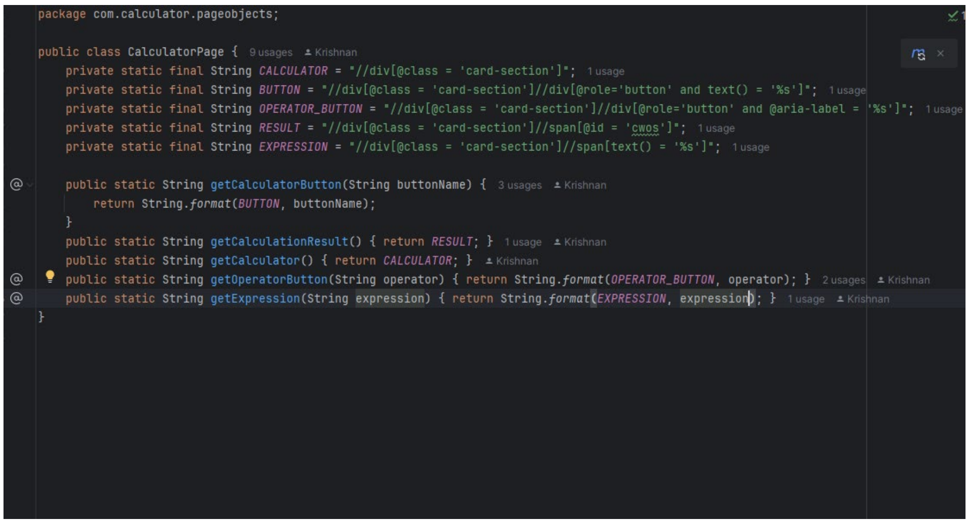
```java
package com.calculator.stepdefinitions;

import ...

public class CalculatorTest {
    private final SeleniumWrapper SELENIUM_UTIL;
    public CalculatorTest(DriverFactory driverFactory) {
        this.SELENIUM_UTIL = new SeleniumWrapper(driverFactory.getDriverInstance());
    }
    @Given("I have opened the Google Calculator")
    public void openGoogleCalculator() {
        SELENIUM_UTIL.sendKeys(By.xpath(GooglePage.getGoogleSearch()), "Calculator");
        SELENIUM_UTIL.sendKeys(By.xpath(GooglePage.getGoogleSearch()), Keys.RETURN);
        Assert.assertTrue(SELENIUM_UTIL.isElementDisplayed(By.xpath(CalculatorPage.getCalculator())), "Calculator is not displayed a
    }
    @When("I press {string}")
    public void pressCalculatorButton(String button) {
      enterDigits(button);
    }
    @When("I press operator {string}")
    public void pressOperatorButton(String button) {
        SELENIUM_UTIL.click(By.xpath(CalculatorPage.getOperatorButton(CalculatorUtils.getOperatorSymbol(button))));
    }
    @When("^I (?:press|click) (?:the ) ?(AC|CE) button$")
    public void pressACorCEButton(String button) {
        SELENIUM_UTIL.click(By.xpath(CalculatorPage.getCalculatorButton(button)));
    }
```

Calculator.feature



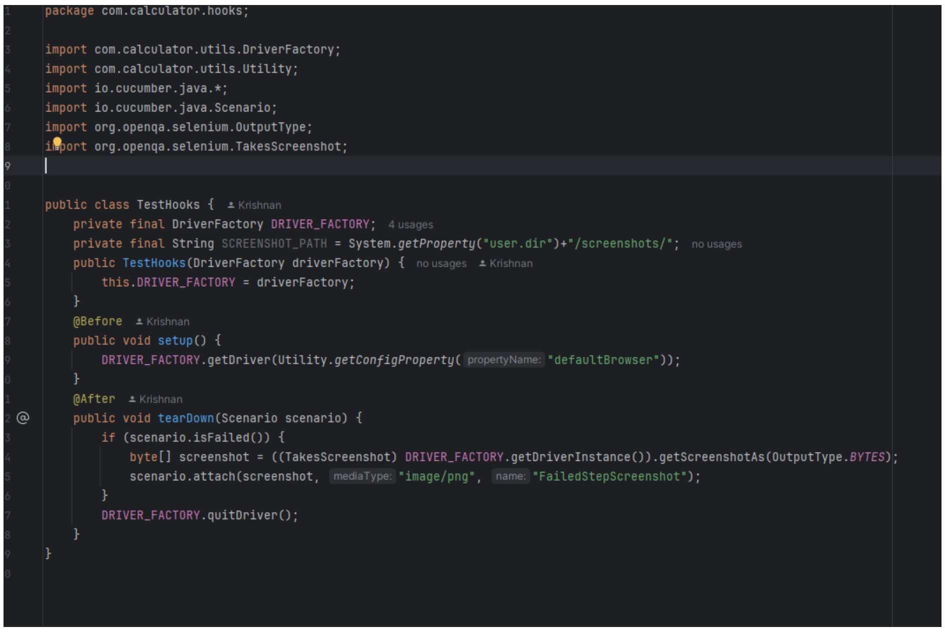
```gherkin
@Regression
Feature: Google Calculator
  This feature is for verifying the functionalities of google calculator

  Background:
    Given I have opened the Google Calculator

  Scenario: Verify functionality of all calculator buttons
    When I press "12345"
    Then the result should be "12345" and the expression should be "Ans = 0"
    And I press operator "+"
    And I press "67"
    Then the result should be "12345 + 67" and the expression should be "Ans = 0"
    And I press operator "-"
    And I press "8"
    Then the result should be "12345 + 67 - 8" and the expression should be "Ans = 0"
    And I press operator "*"
    And I press "9"
    Then the result should be "12345 + 67 - 8 × 9" and the expression should be "Ans = 0"
    And I press operator "/"
    And I press "0.0"
    Then the result should be "12345 + 67 - 8 × 9 ÷ 0.0" and the expression should be "Ans = 0"

  Scenario Outline: Verify basic arithmetic calculation is working as expected
    When I press "<operand1>" "<operator>" "<operand2>" "="
    Then the result should be "<result>" and the expression should be "<expression>"
    Examples:
      | operand1   | operator | operand2   | result        | expression                  |
      | 2          | +        | 3          | 5             | 2 + 3 =                     |
      | 123456789  | -        | 123456789  | 0             | 123456789 - 123456789 =     |
      | 9999999999 | *        | 9999999999 | 1e+20         | 9999999999 × 9999999999 = |
      | 12345      | /        | 6789       | 1.81838267786 | 12345 ÷ 6789 =              |
      | 0.25       | +        | 2.50       | 2.75          | 0.25 + 2.50 =               |
```

CalculatorPage.java



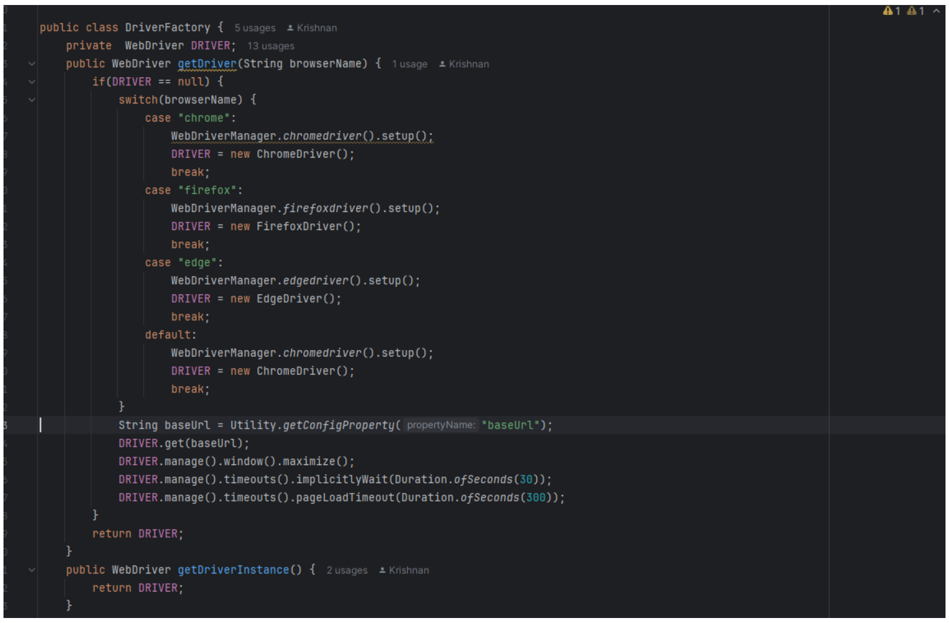
```java
package com.calculator.pageobjects;

public class CalculatorPage {
    private static final String CALCULATOR = "//div[@class = 'card-section']";
    private static final String BUTTON = "//div[@class = 'card-section']//div[@role='button' and text() = '%s']";
    private static final String OPERATOR_BUTTON = "//div[@class = 'card-section']//div[@role='button' and @aria-label = '%s']";
    private static final String RESULT = "//div[@class = 'card-section']//span[@id = 'cwos']";
    private static final String EXPRESSION = "//div[@class = 'card-section']//span[text() = '%s']";

    public static String getCalculatorButton(String buttonName) {
        return String.format(BUTTON, buttonName);
    }
    public static String getCalculationResult() { return RESULT; }
    public static String getCalculator() { return CALCULATOR; }
    public static String getOperatorButton(String operator) { return String.format(OPERATOR_BUTTON, operator); }
    public static String getExpression(String expression) { return String.format(EXPRESSION, expression); }
}
```

TestHooks.java



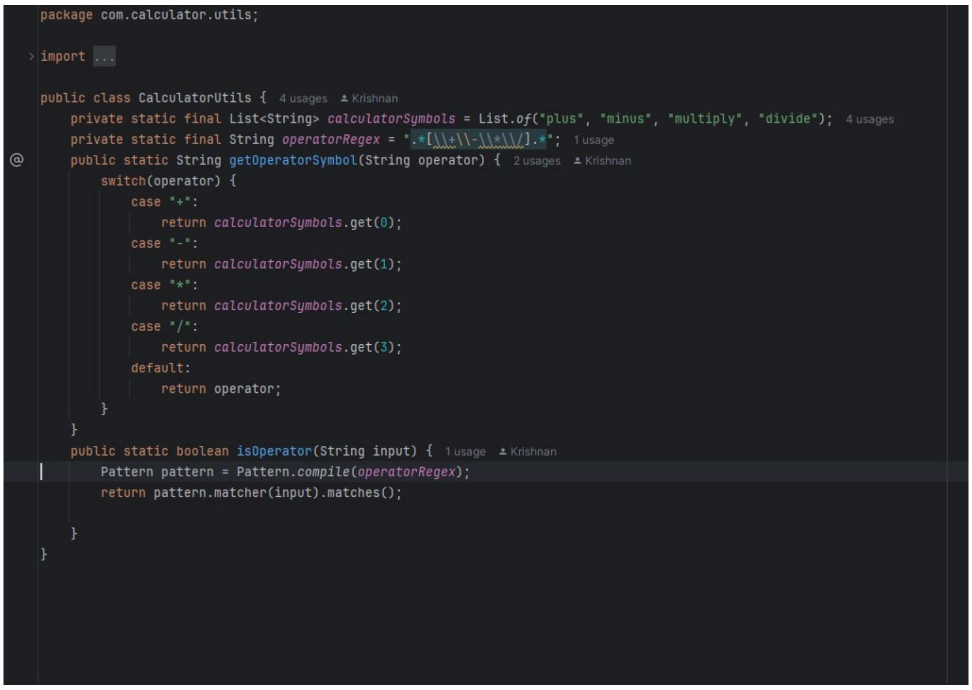
```java
package com.calculator.hooks;

import com.calculator.utils.DriverFactory;
import com.calculator.utils.Utility;
import io.cucumber.java.*;
import io.cucumber.java.Scenario;
import org.openqa.selenium.OutputType;
import org.openqa.selenium.TakesScreenshot;


public class TestHooks {
    private final DriverFactory DRIVER_FACTORY;
    private final String SCREENSHOT_PATH = System.getProperty("user.dir")+"/screenshots/";
    public TestHooks(DriverFactory driverFactory) {
        this.DRIVER_FACTORY = driverFactory;
    }
    @Before
    public void setup() {
        DRIVER_FACTORY.getDriver(Utility.getConfigProperty("defaultBrowser"));
    }
    @After
    public void tearDown(Scenario scenario) {
        if (scenario.isFailed()) {
            byte[] screenshot = ((TakesScreenshot) DRIVER_FACTORY.getDriverInstance()).getScreenshotAs(OutputType.BYTES);
            scenario.attach(screenshot, "image/png", "FailedStepScreenshot");
        }
        DRIVER_FACTORY.quitDriver();
    }
}
```

DriverFactory.java



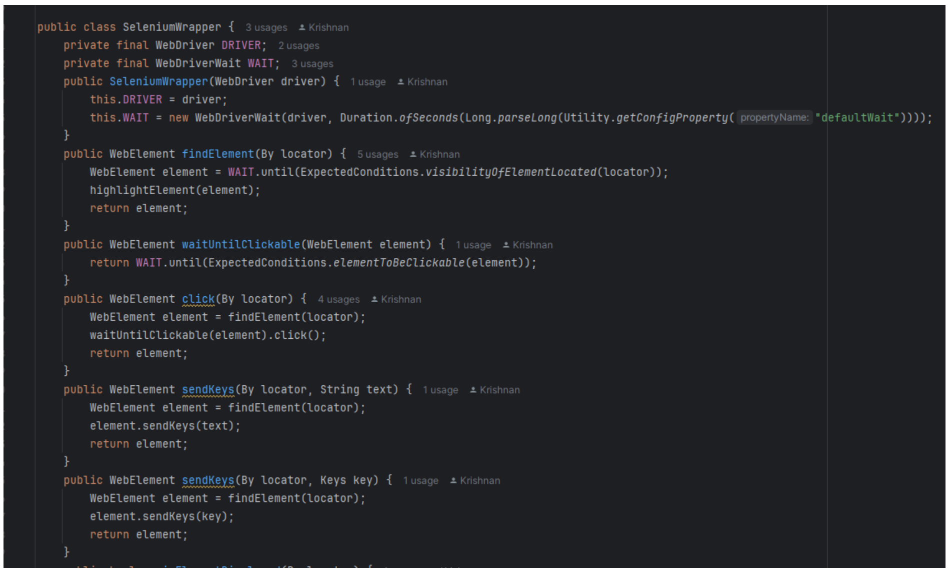
```java
public class DriverFactory {
    private  WebDriver DRIVER;
    public WebDriver getDriver(String browserName) {
        if(DRIVER == null) {
            switch(browserName) {
                case "chrome":
                    WebDriverManager.chromedriver().setup();
                    DRIVER = new ChromeDriver();
                    break;
                case "firefox":
                    WebDriverManager.firefoxdriver().setup();
                    DRIVER = new FirefoxDriver();
                    break;
                case "edge":
                    WebDriverManager.edgedriver().setup();
                    DRIVER = new EdgeDriver();
                    break;
                default:
                    WebDriverManager.chromedriver().setup();
                    DRIVER = new ChromeDriver();
                    break;
            }
            String baseUrl = Utility.getConfigProperty("baseUrl");
            DRIVER.get(baseUrl);
            DRIVER.manage().window().maximize();
            DRIVER.manage().timeouts().implicitlyWait(Duration.ofSeconds(30));
            DRIVER.manage().timeouts().pageLoadTimeout(Duration.ofSeconds(300));
        }
        return DRIVER;
    }
    public WebDriver getDriverInstance() {
        return DRIVER;
    }
```

CalculatorUtil.java



```java
package com.calculator.utils;

import ...

public class CalculatorUtils {  4 usages  Krishnan
    private static final List<String> calculatorSymbols = List.of("plus", "minus", "multiply", "divide");  4 usages
    private static final String operatorRegex = ".*[\\+\\-\\*\\/].*";  1 usage
    public static String getOperatorSymbol(String operator) {  2 usages  Krishnan
        switch(operator) {
            case "+":
                return calculatorSymbols.get(0);
            case "-":
                return calculatorSymbols.get(1);
            case "*":
                return calculatorSymbols.get(2);
            case "/":
                return calculatorSymbols.get(3);
            default:
                return operator;
        }
    }
    public static boolean isOperator(String input) {  1 usage  Krishnan
        Pattern pattern = Pattern.compile(operatorRegex);
        return pattern.matcher(input).matches();

    }
}
```

SeleniumWrapper.java

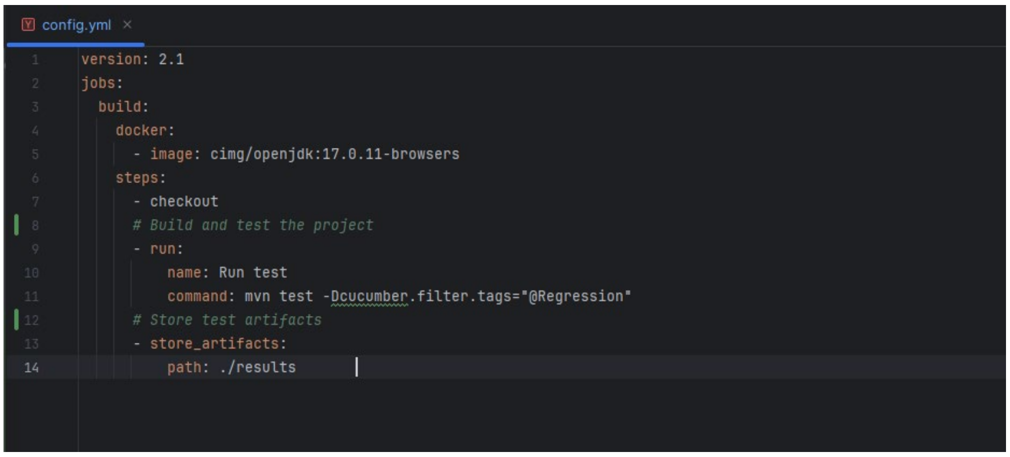


```java
public class SeleniumWrapper {  3 usages  Krishnan
    private final WebDriver DRIVER;  2 usages
    private final WebDriverWait WAIT;  3 usages
    public SeleniumWrapper(WebDriver driver) {  1 usage  Krishnan
        this.DRIVER = driver;
        this.WAIT = new WebDriverWait(driver, Duration.ofSeconds(Long.parseLong(Utility.getConfigProperty( propertyName: "defaultWait"))));
    }
    public WebElement findElement(By locator) {  5 usages  Krishnan
        WebElement element = WAIT.until(ExpectedConditions.visibilityOfElementLocated(locator));
        highlightElement(element);
        return element;
    }
    public WebElement waitUntilClickable(WebElement element) {  1 usage  Krishnan
        return WAIT.until(ExpectedConditions.elementToBeClickable(element));
    }
    public WebElement click(By locator) {  4 usages  Krishnan
        WebElement element = findElement(locator);
        waitUntilClickable(element).click();
        return element;
    }
    public WebElement sendKeys(By locator, String text) {  1 usage  Krishnan
        WebElement element = findElement(locator);
        element.sendKeys(text);
        return element;
    }
    public WebElement sendKeys(By locator, Keys key) {  1 usage  Krishnan
        WebElement element = findElement(locator);
        element.sendKeys(key);
        return element;
    }
```

CircleCI Pipeline Configuration

The CircleCI pipeline is configured to:

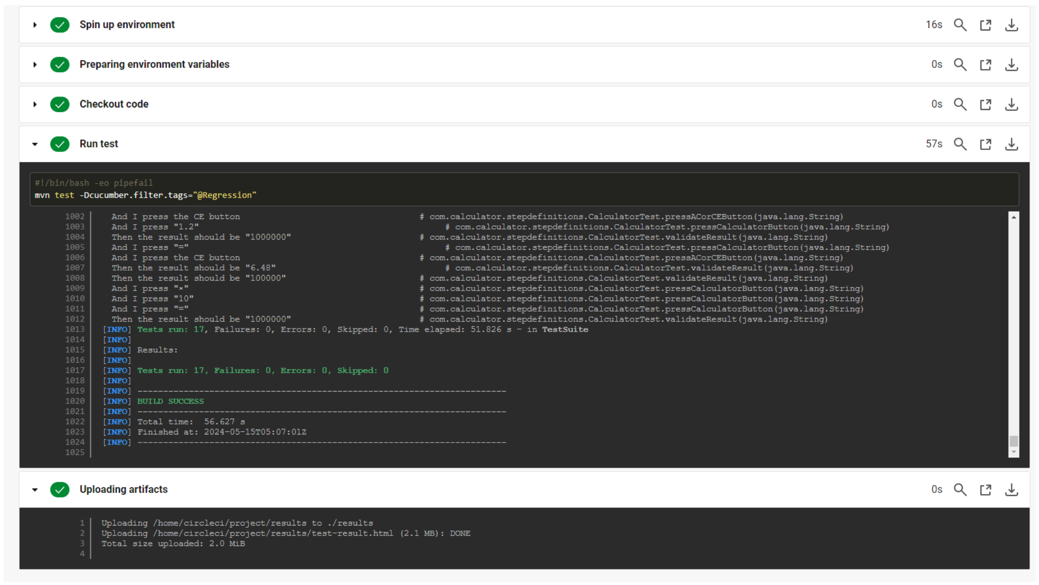
- Build the project.
- Execute automated tests using Maven.
- Generate test reports.
- Publish test results.



config.yml

```yaml
version: 2.1
jobs:
  build:
    docker:
      - image: cimg/openjdk:17.0.11-browsers
    steps:
      - checkout
      # Build and test the project
      - run:
          name: Run test
          command: mvn test -Dcucumber.filter.tags="@Regression"
      # Store test artifacts
      - store_artifacts:
          path: ./results
```

CircleCI Execution Results:



Spin up environment 16s

Preparing environment variables 0s

Checkout code 0s

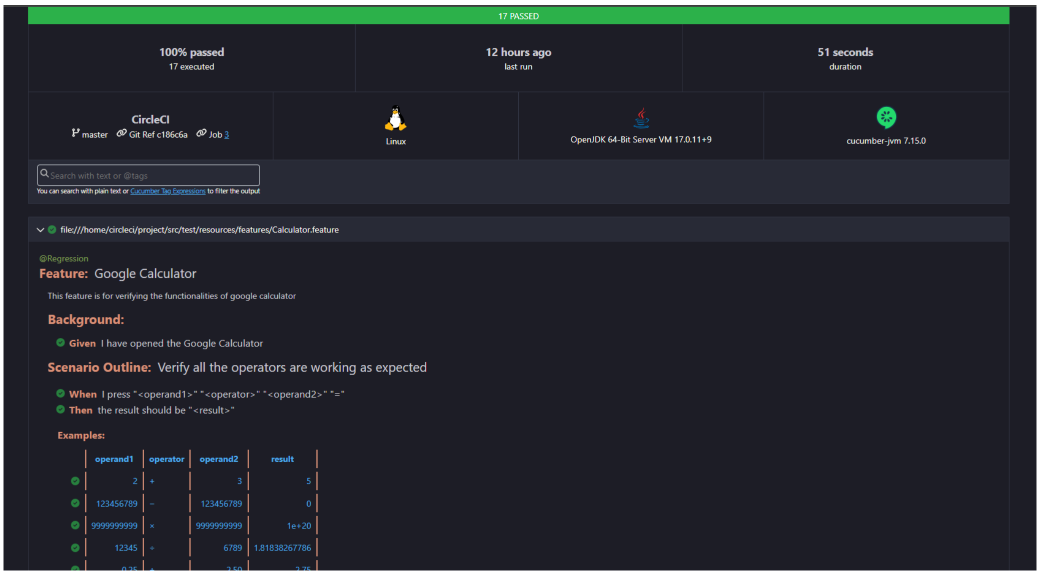
Run test 57s

```
#!/bin/bash -eo pipefail
mvn test -Dcucumber.filter.tags="@Regression"
```

```
   And I press the CE button                              # com.calculator.stepdefinitions.CalculatorTest.pressACorCEButton(java.lang.String)
   And I press "1.2"                                          # com.calculator.stepdefinitions.CalculatorTest.pressCalculatorButton(java.lang.String)
   Then the result should be "1000000"                    # com.calculator.stepdefinitions.CalculatorTest.validateResult(java.lang.String)
   And I press "="                                            # com.calculator.stepdefinitions.CalculatorTest.pressCalculatorButton(java.lang.String)
   And I press the CE button                              # com.calculator.stepdefinitions.CalculatorTest.pressACorCEButton(java.lang.String)
   Then the result should be "6.48"                           # com.calculator.stepdefinitions.CalculatorTest.validateResult(java.lang.String)
   Then the result should be "100000"                     # com.calculator.stepdefinitions.CalculatorTest.validateResult(java.lang.String)
   And I press "×"                                        # com.calculator.stepdefinitions.CalculatorTest.pressCalculatorButton(java.lang.String)
   And I press "10"                                       # com.calculator.stepdefinitions.CalculatorTest.pressCalculatorButton(java.lang.String)
   And I press "="                                        # com.calculator.stepdefinitions.CalculatorTest.pressCalculatorButton(java.lang.String)
   Then the result should be "1000000"                    # com.calculator.stepdefinitions.CalculatorTest.validateResult(java.lang.String)
[INFO] Tests run: 17, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 51.826 s - in TestSuite
[INFO]
[INFO] Results:
[INFO]
[INFO] Tests run: 17, Failures: 0, Errors: 0, Skipped: 0
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  56.627 s
[INFO] Finished at: 2024-05-15T05:07:01Z
[INFO] ------------------------------------------------------------------------
```

Uploading artifacts 0s

```
Uploading /home/circleci/project/results to ./results
Uploading /home/circleci/project/results/test-result.html (2.1 MB): DONE
Total size uploaded: 2.0 MiB
```



17 PASSED

100% passed
17 executed

12 hours ago
last run

51 seconds
duration

CircleCI
master Git Ref c186c6a Job 3

Linux

OpenJDK 64-Bit Server VM 17.0.11+9

cucumber-jvm 7.15.0

Search with text or @tags
You can search with plain text or Cucumber Tag Expressions to filter the output

file:///home/circleci/project/src/test/resources/features/Calculator.feature

@Regression
Feature: Google Calculator

This feature is for verifying the functionalities of google calculator

Background:

✓ **Given** I have opened the Google Calculator

Scenario Outline: Verify all the operators are working as expected

✓ **When** I press "<operand1>" "<operator>" "<operand2>" "="
✓ **Then** the result should be "<result>"

Examples:

	operand1	operator	operand2	result
✓	2	+	3	5
✓	123456789	−	123456789	0
✓	9999999999	×	9999999999	1e+20
✓	12345	÷	6789	1.81838267786
✓	0.25	+	2.50	2.75

Reasoning

1. **Choice of Technologies:** Java was chosen as the primary programming language due to its popularity, strong community support, and extensive libraries for test automation. Selenium WebDriver was selected for browser automation, and Cucumber was used for behavior-driven development (BDD) to enhance collaboration between stakeholders. TestNG was chosen for test execution and reporting due to its robustness and integration capabilities with Maven.
2. **Framework Design:** The framework follows a modular design pattern to ensure maintainability, scalability, and reusability of code. Separation of concerns is achieved by organizing code into packages based on functionality (e.g., step definitions, hooks, utilities). The use of hooks allows for easy management of setup and teardown activities, while the inclusion of utility classes centralizes common functionalities, promoting code reuse. The use of a Selenium wrapper class abstracts WebDriver interactions. Parallel execution is configured to improve test execution time and efficiency.
3. **Reporting:** Cucumber's built-in reporting capabilities provide comprehensive HTML reports with detailed information about test results, including passed, failed, and skipped tests. Screenshots are included in the reports to aid in debugging and troubleshooting failed scenarios, enhancing the visibility of issues and facilitating quicker resolution.

Conclusion

The test automation framework developed provides a robust and scalable solution for UI testing of the calculator application. By leveraging Java, Selenium, Cucumber, and TestNG, the framework enables efficient test creation, execution, and reporting, ultimately improving the quality and reliability of the tested software. The support for parallel execution further enhances the framework's performance and efficiency.