**1. Automating API Testing with Groovy & RestAssured**

**Overview:**

This project will automate API testing using Groovy and the RestAssured library. It will send HTTP requests, validate responses, and generate a test report.

**Tech Stack:**

* Groovy
* RestAssured
* Gradle
* JUnit (for test reporting)

**Steps for Implementation:**

1. **Set Up Your Groovy Project**:
   * First, create a Gradle project and apply the necessary dependencies.
   * Add Groovy and RestAssured dependencies to your build.gradle file.

groovy

Copy

plugins {

id 'groovy'

id 'java'

}

repositories {

mavenCentral()

}

dependencies {

implementation 'io.rest-assured:rest-assured:4.5.1'

testImplementation 'org.codehaus.groovy:groovy-all:3.0.8'

testImplementation 'org.junit.jupiter:junit-jupiter-api:5.7.2'

testImplementation 'org.junit.jupiter:junit-jupiter-engine:5.7.2'

}

1. **Create a Groovy Test Class**:
   * Create a Groovy class (APITest.groovy) to define the tests for different API endpoints.

groovy

Copy

import io.restassured.RestAssured

import io.restassured.response.Response

import org.junit.jupiter.api.Test

import static io.restassured.RestAssured.given

import static org.hamcrest.Matchers.\*

class APITest {

@Test

void testGetUserDetails() {

Response response = given()

.baseUri("https://jsonplaceholder.typicode.com")

.basePath("/users/1")

.when()

.get()

// Validate the response

response.then()

.statusCode(200)

.body("name", equalTo("Leanne Graham"))

.body("email", containsString("@"))

}

@Test

void testCreateNewUser() {

def newUser = [

name: "John Doe",

username: "johndoe",

email: "johndoe@example.com"

]

Response response = given()

.baseUri("https://jsonplaceholder.typicode.com")

.basePath("/users")

.contentType("application/json")

.body(newUser)

.when()

.post()

// Validate the response

response.then()

.statusCode(201)

.body("name", equalTo("John Doe"))

.body("email", equalTo("johndoe@example.com"))

}

}

1. **Run Tests with Gradle**:
   * Use Gradle to run the tests.

bash

Copy

./gradlew test

1. **Generate Test Reports**:
   * Gradle can automatically generate test reports. You can find the test results in the build/reports/tests/test directory.
2. **Enhancements**:
   * Add additional validation for headers, response times, and error conditions.
   * Implement a BDD-style approach with **Cucumber** for more complex test scenarios.

**Documentation:**

**Title**: Automating API Testing with Groovy & RestAssured

**Objective**: This project aims to automate the testing of REST APIs by sending HTTP requests and validating various response properties, such as status code, response body, and headers.

**Steps**:

1. Set up a Gradle project with necessary dependencies.
2. Write test cases using RestAssured for API requests.
3. Validate the responses and ensure that the status codes and content match expectations.
4. Run the tests and generate reports for the results.

**Use Cases**:

* **API Testing**: Validate APIs in a development or CI/CD pipeline.
* **Automated QA**: Automatically run API tests whenever code changes occur.

**2. Web Scraping with Groovy & Jsoup**

**Overview:**

This project will scrape data from a website using Groovy and Jsoup, then store the extracted data into a CSV file.

**Tech Stack:**

* Groovy
* Jsoup
* CSV Parsing

**Steps for Implementation:**

1. **Set Up Your Groovy Project**:
   * First, create a Groovy project and include the necessary dependencies.

groovy

Copy

repositories {

mavenCentral()

}

dependencies {

implementation 'org.jsoup:jsoup:1.14.3'

implementation 'org.apache.commons:commons-csv:1.9.0'

}

1. **Write the Web Scraping Script**:
   * Create a Groovy script (WebScraper.groovy) to scrape data from a website. For demonstration, we’ll scrape basic information like titles from a news site.

groovy

Copy

@Grab(group='org.jsoup', module='jsoup', version='1.14.3')

@Grab(group='org.apache.commons', module='commons-csv', version='1.9.0')

import org.jsoup.Jsoup

import org.jsoup.nodes.Document

import org.apache.commons.csv.CSVFormat

import org.apache.commons.csv.CSVPrinter

import java.nio.file.Files

import java.nio.file.Paths

class WebScraper {

static void main(String[] args) {

def url = "https://news.ycombinator.com/"

Document doc = Jsoup.connect(url).get()

// Extract titles from the page

def titles = doc.select(".storylink")

def articles = titles.collect { [it.text(), it.attr("href")] }

// Create and write to CSV

def csvFile = Paths.get("scraped\_data.csv").toFile()

def writer = Files.newBufferedWriter(csvFile.toPath())

def csvPrinter = new CSVPrinter(writer, CSVFormat.DEFAULT.withHeader("Title", "URL"))

articles.each { article ->

csvPrinter.printRecord(article)

}

csvPrinter.flush()

csvPrinter.close()

println("Scraping completed and data saved to scraped\_data.csv")

}

}

1. **Run the Script**:
   * Execute the script to scrape data from the website and generate the CSV file.

bash

Copy

groovy WebScraper.groovy

1. **Enhancements**:
   * Implement pagination or scraping multiple pages.
   * Add error handling for connection issues or invalid HTML structure.
   * Enhance the script to scrape other types of data like images, links, etc.

**Documentation:**

**Title**: Web Scraping with Groovy & Jsoup

**Objective**: This project demonstrates how to scrape data from a website using Groovy and the Jsoup library, and save the data into a CSV file for later analysis.

**Steps**:

1. Set up a Groovy project with Jsoup and CSV dependencies.
2. Write a Groovy script that connects to the target website, parses HTML, and extracts relevant information (titles, links, etc.).
3. Save the extracted data into a CSV file for further use or analysis.

**Use Cases**:

* **Data Scraping**: Collect and analyze data from websites (e.g., news, blogs, e-commerce).
* **Automation**: Automate data extraction tasks, such as monitoring product prices, news headlines, or job listings.

**Conclusion:**

Both of these projects serve as great introductions to automation and testing. The API testing project leverages Groovy and RestAssured for a smooth, automated testing process, while the web scraping project demonstrates how to pull information from the web and store it efficiently. Would you like help with any specific part of these projects, or do you need additional features or optimizations?

Top of Form

Bottom of Form