**Integrating ChatGPT with Java, Selenium, and TestNG automation framework**

Imagen que contiene luz

Descripción generada automáticamente

This blog aims to explain how to integrate [ChatGPT](https://chat.openai.com/auth/login) with [Java](https://www.java.com/en/), [Selenium](https://www.selenium.dev/), and [TestNG](https://testng.org/doc/) automation framework for natural language-based automated testing. Readers will better understand ChatGPT and its capabilities and how it can simplify the testing process. The blog also explores the potential of ChatGPT as a low-code solution for testing automation and discusses its limitations.

**Introduction**

As we all know, Artificial Intelligence is the future. Currently, ChatGPT, an OpenAI chatbot/trained model that is still being created and trained by their engineers, is a hot topic worldwide. A conversational AI that can answer questions in plain language, write poems, screenplays, social media posts, descriptive essays, and a tonne of other great stuff. In its first week of service, ChatGPT shattered every Internet record by becoming extremely popular. So I wondered how I could use it to make automation testing efforts easier.

As automation testing may be performed more frequently and requires less time and effort than manual testing, it provides faster feedback. However, automation testing also has challenges, including the requirement for technical expertise to write and maintain test scripts, the high cost of the tools used in automation testing, and the difficulty in developing test cases that are both efficient and accurate.

This is where ChatGPT can potentially be a valuable tool for automation testing. By integrating ChatGPT with automation testing frameworks, test cases can be created using natural language. Additionally, ChatGPT can generate code snippets and guide best practices, saving time and effort in the automation testing process. However, it’s essential to keep in mind the limitations of ChatGPT, such as the need to thoroughly check and test generated code before using it in a project.

**ChatGPT**

ChatGPT is a conversational, natural language processing chatbot that uses the Generative Pre-Trained Transformer (GPT) model. ChatGPT, which is based on GPT-3.5, can communicate using natural English. It is capable of correctly responding to queries in a human-like manner.

ChatGPT is a web application based on an AI model that allows users to interact with OpenAI’s AI naturally and conversationally. It can interpret natural language input, respond to it, and remember previous conversations to give more precise and relevant answers.

To automate processes like code generation and testing, ChatGPT can produce text in a variety of forms, including structured data, code snippets, and annotations.

Here are a few ways in which ChatGPT can be used to learn how to code:

* Code generation is an excellent method for beginners to learn a programming language’s fundamental syntax and structure because it can generate code snippets in response to natural language prompts.
* It can be used to finish partly written code, give instructions on what to do next and assist beginners in understanding how to handle a specific issue.
* It can be used to illustrate the function and meaning of particular lines of code, assisting new programmers in learning how various parts of a program interact.
* For beginners who want to create a project, it can offer direction and recommendations for the project’s organizational structure, best practices, and libraries to use.

**Integration of ChatGPT with a testing framework**

Putting the necessary software and libraries in place is the first step. Java, Selenium, and TestNG must all be installed on your computer.

**Step 1: Pre-requisites**

* [Java](https://www.java.com/en/download/) installed on your system
* [Selenium](https://www.selenium.dev/downloads/) WebDriver
* [TestNG](https://testng.org/doc/download.html) testing framework
* POI library to read/write Excel files
* Access to [OpenAPI](https://platform.openai.com/account/api-keys" \t "_blank) for API Key

**Step 2: Spreadsheet with test step details**

The following step is to create the spreadsheet with the test steps we want to run. We can use any tool, such as Google Sheets or Microsoft Excel. I’m using an Excel sheet as a demonstration tool in this example. Here is a sample of how an Excel sheet could look like:

Tabla

Descripción generada automáticamente

Spreadsheet with the prompt information and test case names

**Step 3: Code to read details from the spreadsheet**

Create a method that will read the test steps from the Excel sheet. Here is an example of how you can read test steps using the Apache POI library:

public class ExcelReader {  
 private List<String> testSteps = new ArrayList<String>();  
 private List<String> testCaseNames = new ArrayList<String>();

public ExcelReader() {  
 // Open the Excel file  
 String currentDirectory = System.getProperty("user.dir");  
 // System.out.println("The current working directory is " + currentDirectory);  
 String filePath = currentDirectory + "/src/main/resources/test-steps.xlsx";  
 // System.out.println("The current working file is " + filePath);  
 FileInputStream file = null;  
 XSSFWorkbook workbook = null;  
 try {  
 file = new FileInputStream(new File(filePath));  
 workbook = new XSSFWorkbook(file);  
 } catch (Exception e) {  
 throw new RuntimeException(e);  
 }  
 XSSFSheet sheet = workbook.getSheetAt(0);  
 // Iterate through the rows and columns of the sheet to read the test case names  
 for (Row row : sheet) {  
 Cell cell = row.getCell(0);  
 String testStep = cell.getStringCellValue();  
 testCaseNames.add(testStep);  
 }  
 // Iterate through the rows and columns of the sheet to read the test steps  
 for (Row row : sheet) {  
 Cell cell = row.getCell(1);  
 String testStep = cell.getStringCellValue();  
 testSteps.add(testStep);  
 }  
 // Close the workbook and file  
 try {  
 workbook.close();  
 file.close();  
 } catch (IOException e) {  
 throw new RuntimeException(e);  
 }  
 } public List<String> readTestSteps() throws Exception {  
 return testSteps;  
 } public List<String> readTestCaseNames() throws Exception {  
 return testCaseNames;  
 }

This Java program called ExcelReader reads data from the Excel file test-steps.xlsx. Two private instance variables called testSteps and testCaseNames are lists of Strings used to hold the data read from the Excel file.

The class constructor initializes the instance variables by reading the Excel file and populating the lists with the test steps and test case names. The first step is to get the current directory using the System.getProperty() method and then append the Excel file name to create the file path. The FileInputStream and XSSFWorkbook classes are then used to read the Excel file.

The code then iterates through each row in the Excel file’s first sheet and retrieves each row’s first and second columns. The first column contains the test case names, which are added to the testCaseNames list, and the second column contains the test steps, which are added to the testSteps list.

The class provides two public methods, readTestSteps() and readTestCaseNames(), that return the populated lists of test steps and test case names, respectively. Other classes can use these methods to retrieve the data from the Excel file.

**Step 4: Create the test files and send a prompt to ChatGPT**

Create a method that generates Java test files based on the test steps read from the Excel sheet. The chatGPTCodeGenerator() method is responsible for accessing the chatGPT APIs to generate the code for the provided prompt and read the response.

Here is an example of how to create Java test files:

public void CreateTestCasesForScenarios() {  
 try {  
 List<String> testSteps = reader.readTestSteps();  
 List<String> testCaseNames = reader.readTestCaseNames();  
 int index =0;  
 for (String testStep : testSteps) {  
 // Create a new test case for the test step  
 String testCaseName = testCaseNames.get(index);

// Generate the code for the test step using the ChatGpt API  
 String generatedCode = GPT.chatGPTCodeGenerator(testStep, testCaseName); // Create a test file  
 String fileName = testCaseName + ".java";  
 String filePath = folderPath + "/" + fileName;  
 File file = new File(filePath);  
 file.getParentFile().mkdirs(); // Create parent directories if they don't exist  
 FileWriter writer = new FileWriter(file); String testCase = generatedCode;  
 writer.write(testCase);  
 System.out.println("Test Case for " + testStep + ": " + testCase);  
 index++;  
 writer.close();  
 }  
 } catch (Exception e) {  
 throw new RuntimeException(e);  
 }

This is a Java method named CreateTestCasesForScenarios() that creates test cases based on the test steps read from an Excel file. The method uses an instance of the ExcelReader class to read the test steps and test case names from the Excel file.

The code iterates through each test step in the testSteps list and generates a new test case for each test step. It uses the testCaseNames list to get the name of the test case corresponding to the current test step using the index variable to keep track of the current position in the list.

Next, the method uses an external API called ChatGPT to generate the code for the test step. The GPT.chatGPTCodeGenerator() method takes in the test step and the name of the test case and generates Java code for the test case. The generated code is stored in a string variable called generatedCode.

Then, the method creates a new Java file for the test case using the name of the test case and writes the generated code to the file. The file is saved to a specified folder using the FileWriter class.

Finally, the method prints out the generated test case to the console for debugging purposes, increments the index variable to move to the next test case, and closes the FileWriter object.

If any exception occurs during the execution of the code, the method throws a RuntimeException.

Here find the code snippet for the chatGPTCodeGenerator() method.

public String chatGPTCodeGenerator(String text, String className) throws Exception {

String apiKey = "Your API\_KEY";  
 String endpointUrl = "https://api.openai.com/v1/completions";  
 String prompt = "Create java, selenium and testNG code to " + text + "Consider " + className + "for class name. You can skip the logic for driver setup and cleaning activity as I already have it created as a part of framework"; HttpURLConnection con = (HttpURLConnection) new URL(endpointUrl).openConnection();  
 con.setRequestMethod("POST");  
 con.setRequestProperty("Content-Type", "application/json");  
 con.setRequestProperty("Authorization", "Bearer "+apiKey); JSONObject data = new JSONObject();  
 data.put("model", "text-davinci-003");  
 data.put("prompt", prompt);  
 data.put("max\_tokens", 4000);  
 data.put("temperature", 1.0);  
 con.setDoOutput(true);  
 con.getOutputStream().write(data.toString().getBytes()); String output = new BufferedReader(new InputStreamReader(con.getInputStream())).lines()  
 .reduce((a, b) -> a + b).get();  
 String generatedCode = new JSONObject(output).getJSONArray("choices").getJSONObject(0).getString("text");  
 return generatedCode;  
 }

When this code is executed, a new file will be created with the name we read from the Excel sheet. This is how the created file will appear. In accordance with how my framework is implemented, I made some changes to the created file. For example, to obtain the driver’s information, I extended the class from the BaseTest class, which contains setup and teardown code.

public class TC\_GithubLoginValidation extends BaseTest{  
 WebDriver driver;  
 @Test  
 public void loginValidation() {  
 driver.get("http://www.github.com");  
 WebElement username = driver.findElement(By.xpath("//input[@id='login\_field']"));  
 username.sendKeys("emailAddress");  
 WebElement password = driver.findElement(By.xpath("//input[@id='password']"));  
 password.sendKeys("Password");  
 WebElement signin = driver.findElement(By.xpath("//input[@value='Sign in']"));  
 signin.click();  
 WebElement errorMessage = driver.findElement(By.className("text-red-600"));  
 Assert.assertEquals(errorMessage.getText(), "Incorrect username or password.");  
 }  
}

**Limitations of ChatGPT for automation testing**

While ChatGPT has a lot of potential to be a low-code solution for automated testing, it still has issues.

* The code generated by ChatGPT might not always function correctly since it does not understand the context or requirements of the task and is also less error-free or optimized. Before using the generated code in any project, checking and testing it thoroughly is crucial.
* Since ChatGPT is a machine-learning model, it can be challenging to understand how it produces particular outputs, making it challenging to hold the model responsible for any errors or inaccuracies.
* ChatGPT cannot find deprecated methods, and there is no practical solution to fix the issue.
* Typically, ChatGPT’s generated code follows standard page structures. Users must therefore identify the problem to request an updated code version.

**Conclusion**

Despite these drawbacks, ChatGPT has tremendous potential as a low-code solution for automation testing. The ChatGPT is a very powerful natural language model with enormous potential. It has tremendous capabilities and will likely become the standard for low-code testing solutions to a wide range of issues. Although it has the ability to do so for testing, using ChatGPT as it stands still requires that the user have a basic understanding of both the language being used and the app being tested. Nonetheless, we shouldn’t underestimate ChatGPT’s potential because it can generate impressive code that was not achievable with earlier natural language generation models.

**References**

[open API platform](https://platform.openai.com/): This will provide detailed information about chatGPT

[automation testing](https://www.tutorialspoint.com/software_testing_dictionary/test_automation.htm): This will give information about test automation