DERIVCO - CLARISSA CROUCH - COCKTAIL ASSESSMENT

Introduction

The Cocktail DB is a public database of cocktails and drinks from around the world https://www.thecocktaildb.com/. The Cocktail DB has a public API to retrieve data about ingredients and drinks. See the documentation here https://www.thecocktaildb.com/api.php. We have proposed some requirements for the cocktail API:

Your task is to:

- → Write a minimum set of test cases to test the requirements below.
- → Write two additional test cases that are not covered by the requirements below.
- → Automate the test cases using a language/framework of your choice.
- → Suggest two non-functional tests that you would design.
- → Suggest a framework that could be used to automate the non-functional test above.
- → Send your tests with instructions on how to execute the automated tests within 3 days.
- → Always explain yourself clearly and let us know if any assumptions were made.

Functional Requirements

1. **Search Ingredients by Name:** www.thecocktaildb.com/api/json/v1/1/search.php?i=vodka

   a. The system shall include a method to search by ingredient name and return the following fields:
      - **Ingredient ID (string),**
      - **Ingredient (string),**
      - **Description (string),**
      - **Type (string),**
      - **Alcohol (string) and**
      - **ABV (string).**

   b. If an ingredient is non-alcoholic, Alcohol is null and ABV is null

   c. If an ingredient is alcoholic, Alcohol is yes and ABV is not null.

2. **Search Cocktails by Name:** www.thecocktaildb.com/api/json/v1/1/search.php?s=margarita

   a. The system shall include a method to search by cocktail name.

   b. If the cocktail does not exist in the cocktail DB, the API shall return drinks as null.

   c. Searching for a cocktail by name is case-insensitive.

   d. API response must contain the following Schema properties: (at the end of the pdf)

Framework suggested to automate the test cases:

- **REST Assured (Java) – this is the one I chose.**

Instructions to execute the automated tests using: Rest Assure Library, Junit3, Java, Maven

- Install Java JDK and set up the environment variables.
- Install Maven and set up the environment variables.
- Create a new Maven project in your preferred IDE.
- Add the following dependencies to the POM.xml file:
  - **io.RestAssured- automate Restful APIs.**
  - **Junit- write and run repeatable automated tests.**
  - **Commons-io- work with files**
  - **Jackson-databind- serialize and deserialize the data.**
  - **Cucumber-junit-**
  - **Cucumber-java-**
  - **Maven-compiler-compile the source code of Maven project**
  - **Json-schema-validator- validate schema of JSON.**
  - **Json-path- extract specific value of JSON.**
  - **Maven-cucumber-reporting- produce reports what scenarios passed or failed.**
- Create a new Java class to write your test cases.
- In the test class, import the required libraries:
  - **import static io.restassured.RestAssured.*;**
  - **import static org.junit.Assert.*;**
  - **import org.junit.Test;**
  - **import io.restassured.response.Response;**
  - **import com.fasterxml.jackson.databind.ObjectMapper;**
- Write your test cases using the Rest Assured syntax and Junit3 assertions.
- Run the tests using Maven. Open a command prompt or terminal, navigate to the project directory, and run the command mvn test.

Non-functional test cases and Framework suggested to automate them:

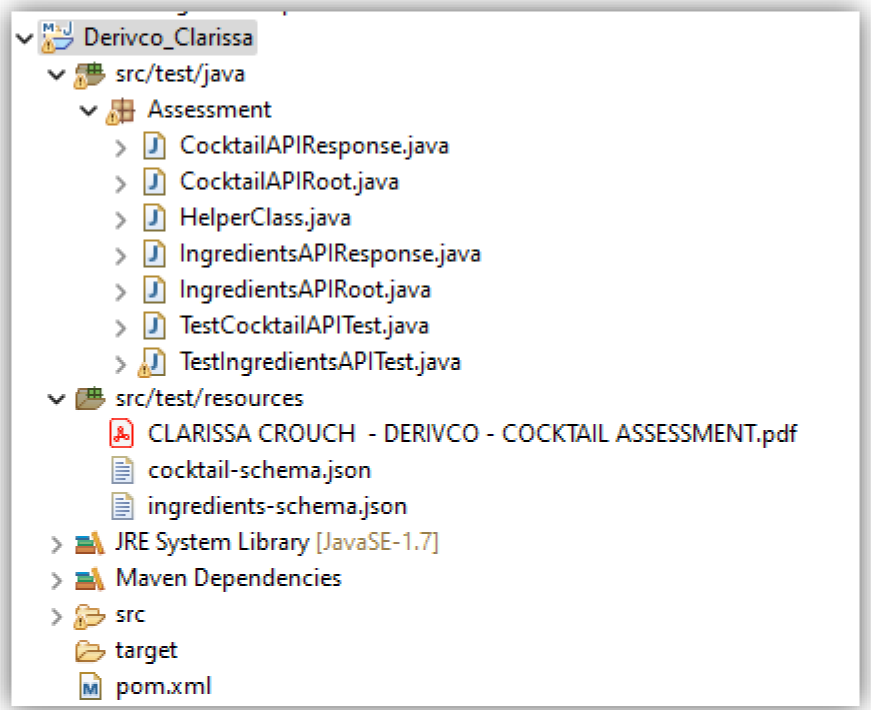
1. <u>**Performance Test:**</u>
   - Simulate a high number of requests to the API.
   - Measure the response time of the API under heavy load.
   - Ensure that the response time is within acceptable limits.

   <u>**Framework**</u>: we can use **JMeter** to simulate heavy load on the API and measure its response time.

2. <u>**Security testing**</u> - test the API's security measures, including authentication and authorization, to ensure that only authorized users can access sensitive data.
   - Attempt to access the API using an incorrect or expired API key.
   - Verify that the API does not allow access and returns an appropriate error message.

   <u>**Framework**</u>: For security tests, we can use a tool like **OWASP ZAP** to attempt to access the API with incorrect or expired API keys and verify that the API behaves as expected.

EXPLAINING MY PROJECT SHARED IN GITHUB:



```
Derivco_Clarissa
  src/test/java
    Assessment
      CocktailAPIResponse.java
      CocktailAPIRoot.java
      HelperClass.java
      IngredientsAPIResponse.java
      IngredientsAPIRoot.java
      TestCocktailAPITest.java
      TestIngredientsAPITest.java
  src/test/resources
    CLARISSA CROUCH  - DERIVCO - COCKTAIL ASSESSMENT.pdf
    cocktail-schema.json
    ingredients-schema.json
  JRE System Library [JavaSE-1.7]
  Maven Dependencies
  src
  target
  pom.xml
```

- **Src/test/java: a package named Assessment:**
  - **Java Class: TestIngredientsAPITest**
  - **Java Class: TestCocktailAPITest**
  - **Java Class: HelperClass**

- **Resources:**
  - **I have the PDF explaining my chain of thought (also sent by email)**
  - **And 2 json files to keep the schema for both Functional Requirements.**

SEARCH INGREDIENTS BY NAME

Test cases to test the requirements: Java Class: TestIngredientsAPITest

1. **Test Case: Test that an API valid search - return a success status code (200).**

   This test case is checking that when a valid search is made on the API, the response status code should be a 200 OK status code, indicating that the API request was successful.

   Explaining the Code - Search: Vodka

   It creates an instance of the ReusableMethods class that I created, which is assumed to have a method searchIngredientName that performs a search for the given ingredient name (String) and returns a Response object. The test then asserts that the response status code is 200 and that the response data is not null, which are appropriate validations for this test case.

2. **Test Case: Test that an API valid search - Schema Validation**

   This test case is to ensure that the API returns responses that conform to the expected schema. Schema validation is a way of ensuring that the structure and data types of the response match a predefined schema. In this test case, we are testing the schema validation of a response from an API search endpoint. The goal of this test case is to ensure that the API returns a response with a structure and data types that conform to the expected schema.

   Explaining the Code - Search: Vodka

   This test case is testing whether the response for searching a valid ingredient by name returns all required fields as per the defined schema in "**ingredients-schema.json**" file. The test case involves sending a request to search for an ingredient by name "Margarita" using a reusable method called "**searchIngredientName**". The response from the API is then validated against a JSON schema using the "**matchesJsonSchemaInClasspath**" method provided by the **JsonSchemaValidator library**.

   - If the response does not match the expected schema, the test will fail.
   - If the response matches the expected schema, the test will pass.

   It's important to note that this test case only validates the schema of the response and does not test the functionality of the API or the accuracy of the returned data.

3. **Test Case: Test that an API valid search - returns all required fields:**

   This test case is checking that when a valid search is made on the API, the response body should contain all the required fields as defined in API's schema. It validates that the API response is returning the expected data.

   Explaining the Code - Search: Vodka

   The test uses the "searchIngredientName" method from the "ReusableMethods" class to search for the ingredient "Vodka" and then validates the response using a JSON schema validator. It then checks that the response contains at least one ingredient and that each ingredient has the required fields "idIngredient", "strIngredient", "strDescription", "strType", "strAlcohol", and "strABV". This helps to ensure that the API is functioning correctly and returning the expected data.

4. Test Case: Test that an API invalid search - returns a null (empty) ingredient array.

This test case is checking that when an **invalid search** is made on the API, and no ingredients are found, the API response should return **in a null or empty ingredient array**. This test verifies that the API returns the expected response when an invalid search is made.

Explaining the Code - Search: Invalid

The test first sends a request to the API using the ReusableMethods class and an invalid search term, "Invalid".
It then verifies that the response status code is 200 and that the response data is not null using assertEquals and **assertNotNull**.
Next, it **extracts the ingredient array from the response** and stores it in **arrayActualSize**.
Finally, it checks that the size of the ingredient array is null, which indicates that the search term did not return any matching ingredients. This is done using assertEquals, where the **expected value is null**, and the **actual value is arrayActualSize**.

---

5. Test Case: Test that an API valid search – if ingredient is "Yes" for Alcohol then: ABV is NOT NULL.

This test case is checking that when a valid search is made on the API for an ingredient that has alcohol, the ABV (Alcohol by Volume) field in the API response **should not be null**.
This test verifies that the API is returning accurate data for ingredients that contain alcohol.

Explaining the Code - Search: Vodka

This code is testing whether an alcoholic ingredient has a non-null strABV field. The test first searches for an ingredient called "Vodka", and then extracts the strAlcohol and strABV fields from the first element in the response array. If the strAlcohol field is "Yes", which indicates that the ingredient is alcoholic, the test asserts that the strABV field is not null. If the strAlcohol field is "No", which indicates that the ingredient is non-alcoholic, the test asserts that the strABV field is null.

---

6. Test Case: Test that an API valid search – if ingredient is "No" for Alcohol then: ABV is NULL.

This test case is checking that when a valid search is made on the API for an ingredient that does not contain alcohol, the ABV field in the API response should be null. This test verifies that the API is returning accurate data for ingredients that do not contain alcohol.

Explaining the Code - Search: water

Same code snippet as Test 5, but different input: "water", expect ABV null.

---

7. Test Case: Test that an API valid search - verify request data is case sensitive ('VoDkA').

This test case is checking that when a valid search is made on the API with a search term that includes upper and lowercase letters, the API response should be case-sensitive and return results matching the exact case of the search term.

Explaining the Code - Search: VoDkA'

This is a test case that checks if the API can properly handle a search query for an ingredient with mixed case.
The test sends a search request for an ingredient with name "vOdKa". The expected result for this search is an ingredient with ID 1, name "Vodka", type "Vodka", an alcohol flag set to "Yes" and ABV of 40.

The test first checks the HTTP status code and the presence of response data. Then it extracts the actual field values from the response using path expressions and compares them with the expected values using assertions. If all assertions pass, the test case passes successfully. **Result: it is not case sensitive.**

SEARCH COCKTAILS BY NAME:

Test cases to test the requirements: Java Class: TestCocktailAPITest

1. **Test case: searching existing cocktail name – verify status code 200, response not null.**

   Verify status code 200: This means that the HTTP status code returned by the server should be 200, which indicates a successful response from the server. A status code other than 200 could indicate an error, such as a missing resource or a server-side issue.
   Verify response not null: This means that the response received from the server should not be null or empty. A null response could indicate a problem with the server, such as a timeout or an error in the search function.

   Explaining the Code - Search: Margarita

   This code is for a test case that checks whether a valid search for a cocktail by name returns a successful status code of 200 and a non-null response.
   It first creates an instance searchByName and calls the searchCocktailName method with the parameter "Margarita" to send a request to search for a cocktail with the given name. Then it uses assertEquals method to check if the status code of the response is 200 and assertNotNull to check if the response data is not null.

---

2. **Test case: searching existing cocktail name - verify drinks array is not null.**

   This test case verifies that when a valid cocktail name is searched, the API returns a non-null "drinks" array in the response. This is important because the "drinks" array contains information about the matching cocktails, and it should not be empty. In this specific test case, the goal is to verify that the response data is not null.

   Explaining the Code - Search: Margarita

   The test creates an object of the ReusableMethods class, which contains a method for searching cocktail by name. It then sends a request to the API to search for "Margarita" and receives a response.
   The assertNotNull method checks that the response object is not null, and then uses the **extract() method to get the response body**, and **path()** method to extract the value of the "drinks" array field from the JSON response.
   If the assertion fails, it means the response does not contain any drinks or the drinks array is null.

---

3. **Test case: searching non-existent cocktail name - verify drinks array is null.**

   This test case verifies that when a non-existent cocktail name is searched, the API returns a null "drinks" array in the response. This is important because the user should receive a clear indication that the cocktail does not exist in the database.

   Explaining the Code - Search: invalid

   This code tests for the **case where an invalid cocktail name is searched** for and verifies that the drink array is null. The code first sends a search request with an invalid cocktail name and then processes the response by mapping it to a CocktailAPIRoot object using the **Jackson ObjectMapper library**. The assertNotNull method is used to check that the response is not null. Then, the code retrieves the drinks array from the response and uses the assertNull method to check that it is null. If the drinks array is not null, the test fails.

---

4. **Test case: searching valid cocktail - verify request data is case sensitive.**

   This test case verifies that when a valid cocktail name is searched, the API is case sensitive and returns a response only if the case of the searched cocktail name matches exactly the case of the cocktail name in the database. This is important because the API should be consistent in its response to different types of requests.

   Explaining the Code - Search: TiPperArY

   This code is implementing a test case to verify if the cocktail search function is case-sensitive. The test case searches for a cocktail with a mixed-case name "TiPperArY" using the searchCocktailName() method from the ReusableMethods class. It then uses the Jackson ObjectMapper to parse the response body as a JSON string and maps it to the CocktailAPIRoot class. The test then checks whether the response status code is 200 using the assertEquals() method. After that, it loops through the list of drinks obtained from the response and checks whether the "TiPperArY" cocktail is present and the response is case sensitive by using assertEquals() method. If it is present, it sets the found flag to true. Finally, it asserts that the flag is false using assertFalse() method indicating that the search is case-sensitive. **Result: it is case sensitive.**

---

5. Test case: searching valid cocktail – Schema Validation

This test case is to ensure that the API returns responses that conform to the expected schema. Schema validation is a way of ensuring that the structure and data types of the response match a predefined schema. In this test case, we are testing the schema validation of a response from an API search endpoint. The goal of this test case is to ensure that the API returns a response with a structure and data types that conform to the expected schema.

Explaining the Code - Search: Margarita
This is a test case that validates the schema of the response received from the search for a cocktail by name. First, it creates an instance of the ReusableMethods class and sends a request to search for the cocktail "Margarita". Then, it uses the JsonSchemaValidator from the REST-assured library to validate the response schema against a JSON schema file called "cocktail-schema.json", which should be located in the classpath. The matchesJsonSchemaInClasspath method checks if the response body matches the schema defined in the specified file. If the response does not match the schema, the test will fail.

---

6. Test case: searching valid cocktail - returns all required fields:

This is another test case for the cocktail search functionality. In this test, the test verifies that the response contains the required fields for each cocktail that matches the search criteria.

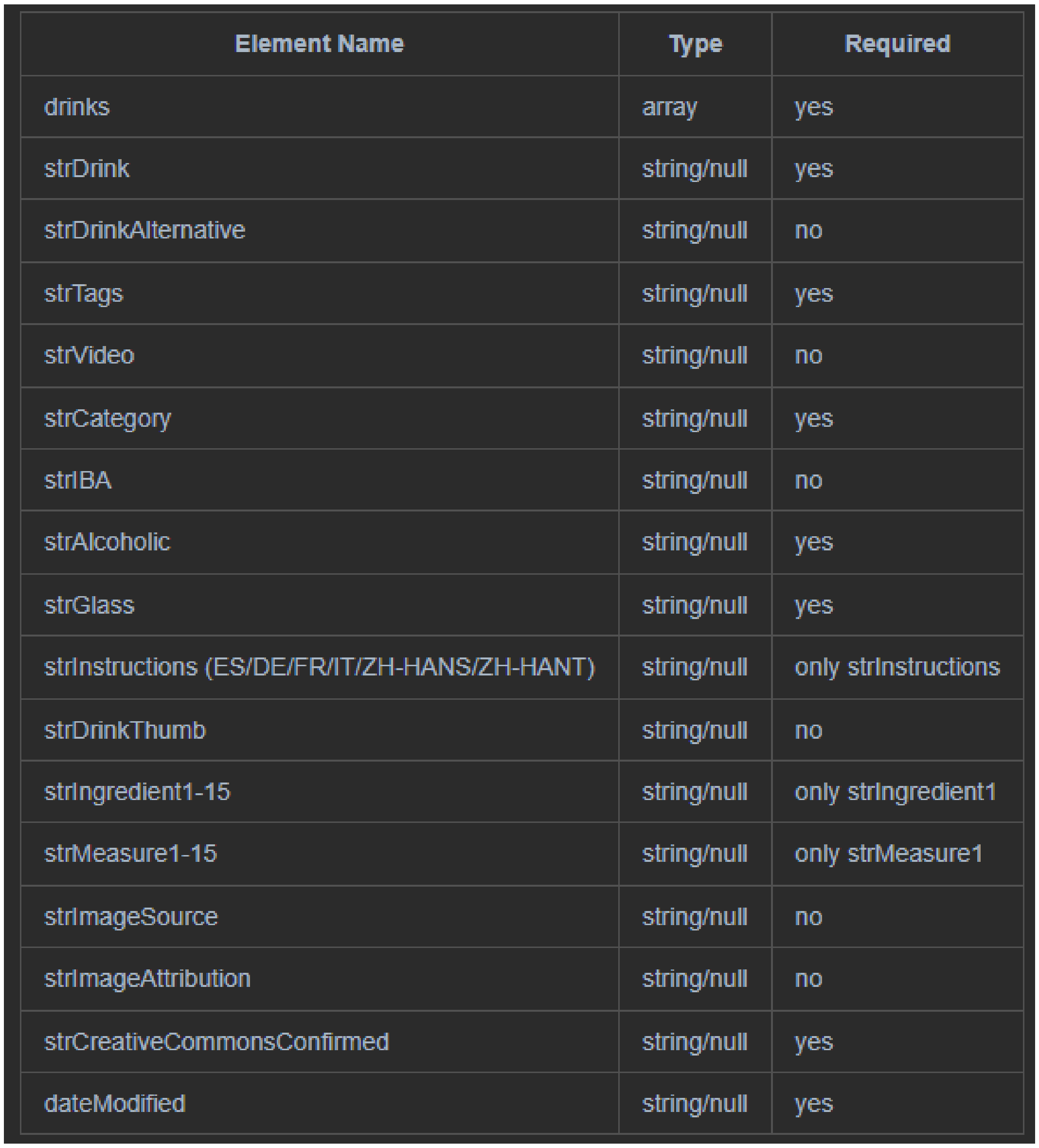
Explaining the Code - Search: Margarita
The test starts by sending a request to search for a cocktail named "Margarita". Then, it validates the response schema using the "cocktail-schema.json" file.
Next, the test verifies that the response contains at least one cocktail.
It then checks that each cocktail has the required fields such as "strDrink", "strTags", "strCategory", "strAlcoholic", "strGlass", "strInstructions", "strIngredient1", "strMeasure1", "strCreativeCommonsConfirmed", and "dateModified". **If any of the required fields is missing, the test fails**.
This test ensures that the response is correctly formatted and includes all the necessary information for each cocktail.

HELPER CLASS:

This is a helper class containing two methods to perform API requests to search for cocktail recipes based on either an **ingredient name** or a **cocktail name**.
The class has two private constants that define the **base URL** and **search path** of the API.
The **searchIngredientName method** takes an ingredient parameter and constructs a URL with the **?i=** query parameter to perform a search by ingredient. The **searchCocktailName method** takes a cocktail parameter and constructs a URL with the **?s=** query parameter to perform a search by cocktail name.

Both methods use the given().when().get() method chain from the RestAssured library to send the HTTP GET requests and return the response. The response is then printed and returned for further processing by the calling code.

API response for Search Cocktails by Name must contain the following Schema properties: (attached)



Element Name	Type	Required
drinks	array	yes
strDrink	string/null	yes
strDrinkAlternative	string/null	no
strTags	string/null	yes
strVideo	string/null	no
strCategory	string/null	yes
strIBA	string/null	no
strAlcoholic	string/null	yes
strGlass	string/null	yes
strInstructions (ES/DE/FR/IT/ZH-HANS/ZH-HANT)	string/null	only strInstructions
strDrinkThumb	string/null	no
strIngredient1-15	string/null	only strIngredient1
strMeasure1-15	string/null	only strMeasure1
strImageSource	string/null	no
strImageAttribution	string/null	no
strCreativeCommonsConfirmed	string/null	yes
dateModified	string/null	yes