Week #6 Homework Part 1

IS 296

Professor Vincent

Jvincent30@stlcc.edu

Due Sunday, July 17th 11:59pm central

(100 points) for Homework, (75 points available for extra credit), Total of 175 points available.

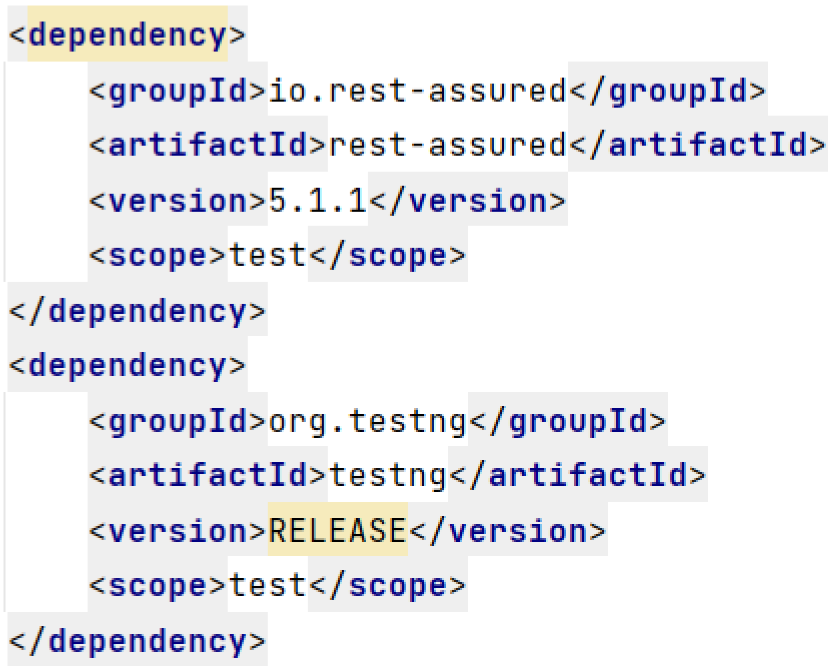
Extra Credit is strongly recommended for this homework.

Topic: Rest Assured

1. This project assumes that you have at least week #4 with the customer review endpoints added to your taco application.
2. Scratch notes regarding some of the testing technology we've been discussing is found at the bottom of this homework document.
3. This project will require you to have 2 instances of intellij open so that you can run one instance with your customer review endpoints via your taco application and a second instance with the tests you're building.
4. So before you continue, go ahead and open your taco application that has the customer review endpoints and run it and make sure it stays up and running with no issues. If there are any issues, make sure those are resolved before you proceed.
5. So, let's keep going and reflect a bit on rest assured, a cool and useful technology, and then proceed to add some hands on experience.
6. The goal here is to increase understanding through both hands on experience and through asking questions about each step that you don't totally understand.
    a. Leverage your tutors, professor, and tech coach to reach out with questions.
7. RestAssured
    a. Is a testing technology
    b. Is very helpful testing API's
    c. Is an alternative approach to Postman
    d. Can integrate with testing frameworks like Junit or TestNg
    e. Is implemented in Groovy
    f. Can create tests using Gherkin terminology
        i. Given, When, Then
    g. Website: REST Assured (rest-assured.io)
        i. Can find many details about RestAssured
    h. Developed and maintained by Johan Haleby
        i. Somebody, like you (after you have much more experience) created this technology
    i. Prerequisites (note: your VM's and STLCC laptops should be good to with this)
        i. Java
            1. We'll assume Java 8
            2. You can validate this by typing java --version

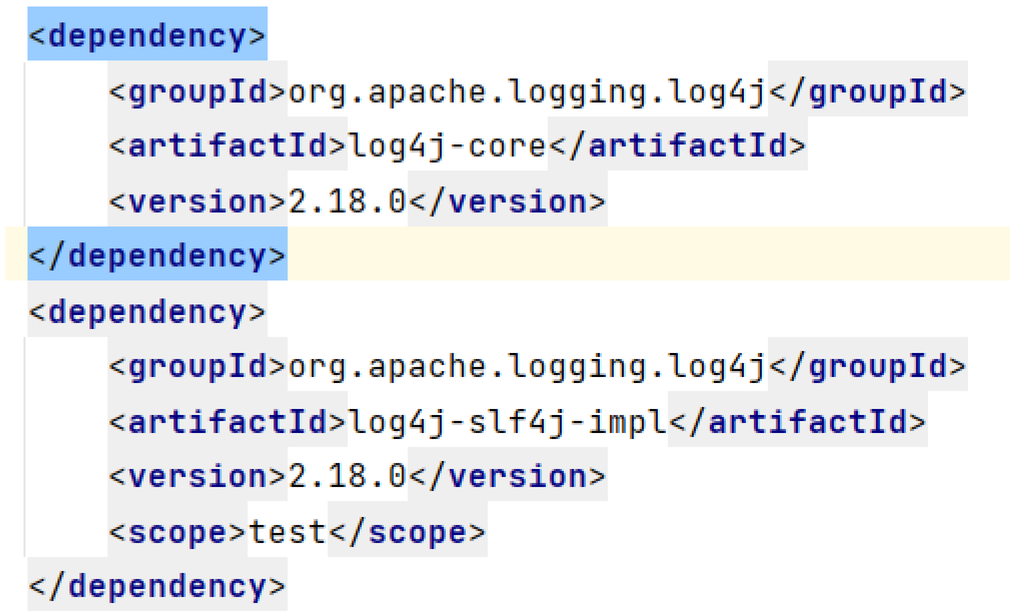
        ii. IDE: i.e. IntelliJ
        iii. Maven
        iv. TestNg: as testing framework
8. Open a second IntelliJ instance. The instructions following apply to the second intellij instance.
9. The second intellij instance will be used to build a small testing application. Yes, there are applications you build that include Java code and tests, and their sole purpose is to test other applications.
10. Reflect on some of the testing discussions we've had in class recently.
    a. TDD
    b. Test Driven Development
        i. Creating Tests first.
        ii. Create code that passes the test second.
    c. Red, Green, Refactor.
        i. Red: write a test that fails.
        ii. Write a test that does not pass.
        iii. Green: write code that allows the test to pass.
        iv. Write only enough code to get the test to pass.
        v. Refactor: improve the code, and make sure it still passes.
        vi. Make sure the code meets team standards.
        vii. Make sure the code is organized, indented well, readable, documented.
        viii. Make sure that it still works.
    d. Benefits:
        i. Often, if you don't write tests up front, they don't get written at all.
        ii. They provide the opportunity to clarify exactly what the requirements are at the beginning, so that there are no surprises late... or at least, as few surprises as possible.
        iii. Confidence, confidence, confidence... because... you know that your work, works... before you tell your team or tech lead that you're done and that your code works.
        iv. Regression testing and confidence: you know that code you wrote a year ago, still works... without taking more and more of your time to manually test it and make sure it works.
    e. Gherkin
        i. Given When Then
    f. Given a situation
    g. When I take one or more actions
    h. Then I assert upon an expected result
    i. Arrange, Act, Assert
        i. Arrange
        ii. Set up the variables and objects you need for a test.
        iii. Act
        iv. Call methods and/or make requests, etc
        v. Assert
        vi. Make sure that results are exactly what you expected them to be

    j. Expect, Measure, Reward
        i. Expect
        ii. Expect some outcome
        iii. Measure
        iv. Measure to see if the outcome happened
        v. Reward
        vi. Provide a passed test if all works as it should.
        vii. Provide a failed test if any part does not work as it should.
11. Challenge yourself to recall what we discussed around those notes above.
12. Ok, now let's proceed with building our small testing application in our second intellij instance.
13. Start a new maven project by completing the sub-steps below.
    a. File -> New -> Project
    b. Maven
        i. Project SDK: 17 version 17.0.1
    c. Click Next
    d. For the name enter: VincentRestassuredProject
        i. Note: use your last name instead
    e. Change the location to be the directory where you want your project to be located.
    f. Click Finish
    g. Click on appropriate decisions for This Window and Trust Project if prompted
14. Remember how we've discussed that maven is a powerful technology that we've used so far to help manage dependencies for our spring boot projects, but that can be used without spring boot too, just like we're doing with this separate testing project we're building.
15. Go to maven central by going to the following website in your browser:
    a. https://mvnrepository.com/repos/central
16. Search for "rest assured" in the maven central search bar
17. Locate the option that has its source as "io.rest-assured" and click on it.
18. Click on the latest version (i.e. 5.1.1 or newer)
19. Click on the maven tab if it's not already selected.
20. Copy the maven dependency
21. Locate and Open your pom.xml
22. Add a dependencies block that will allow you to pull in the RestAssured technology by adding the rest assured dependency that you grabbed from maven central.
23. Go back to maven central.
24. Search for testng. (not testing… testng)
25. Select the option from org.testng
26. Click on the latest version (i.e. 7.6.1 or newer)
27. Add a dependency for testng to your pom.xml.
28. Locate (in maven central) and Add dependencies for log4j and slf4j bridge
29. Your final pom.xml should have a dependencies section (remember, opening and closing tag needed for dependencies i.e. <dependencies></dependencies>) and include the dependencies similar or equal to the below:



```xml
<dependency>
    <groupId>io.rest-assured</groupId>
    <artifactId>rest-assured</artifactId>
    <version>5.1.1</version>
    <scope>test</scope>
</dependency>
<dependency>
    <groupId>org.testng</groupId>
    <artifactId>testng</artifactId>
    <version>RELEASE</version>
    <scope>test</scope>
</dependency>
```

a.



```xml
<dependency>
    <groupId>org.apache.logging.log4j</groupId>
    <artifactId>log4j-core</artifactId>
    <version>2.18.0</version>
</dependency>
<dependency>
    <groupId>org.apache.logging.log4j</groupId>
    <artifactId>log4j-slf4j-impl</artifactId>
    <version>2.18.0</version>
    <scope>test</scope>
</dependency>
```

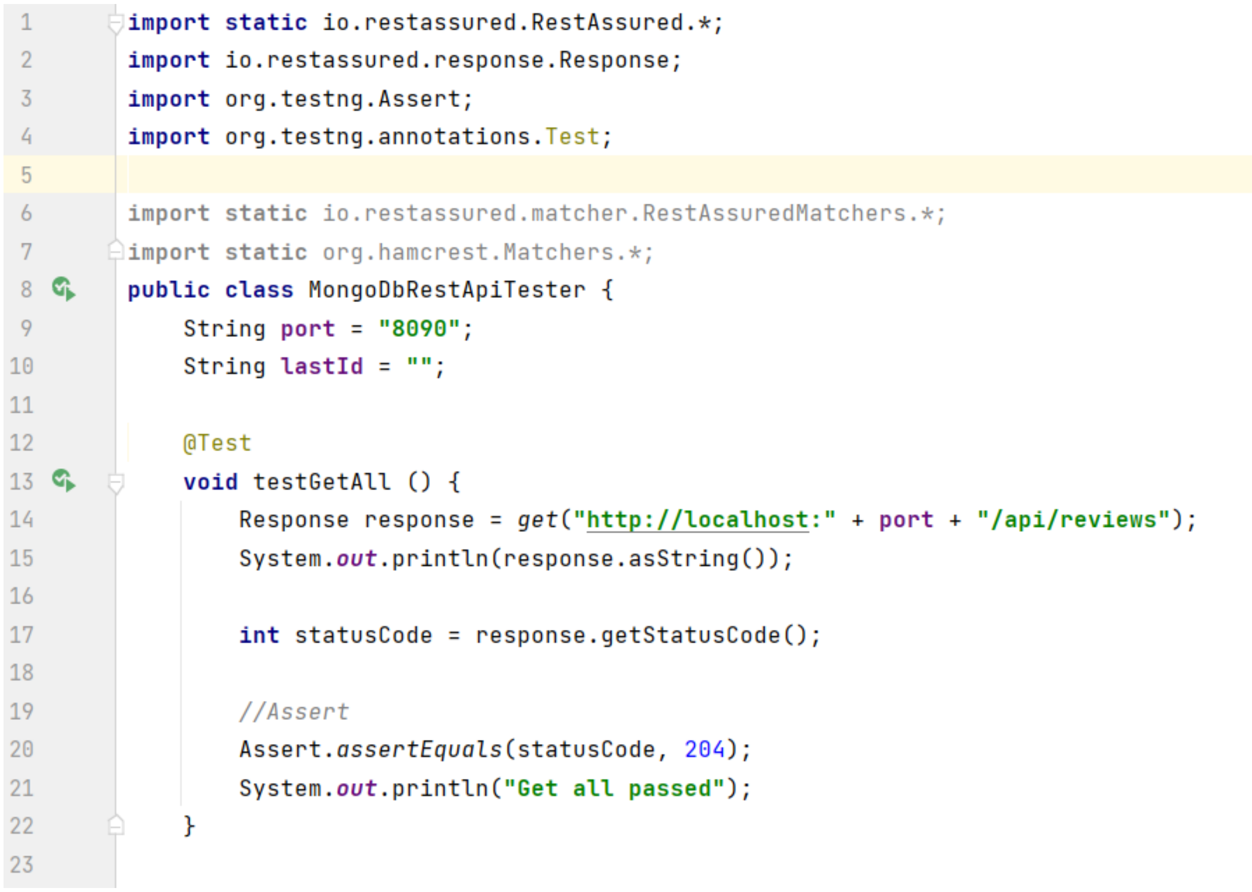
b.

c. Make sure all of the above is between the “dependencies” open and closing tags

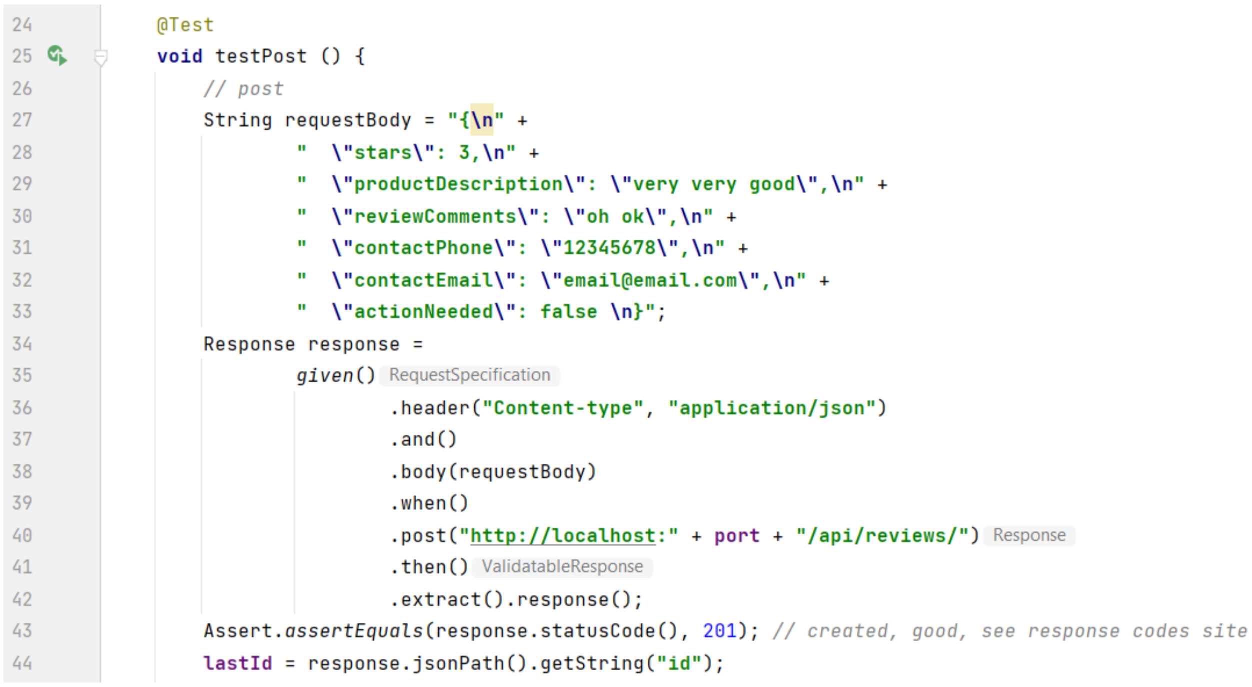
30. Do a bit of research to explore the technologies involved with these dependencies more. Spend a few minutes on each of these please so that you can describe more about them. Notice the “related books” section near the bottom of the maven central link for each one. Here are a sample of links. You want to build a habit of researching the key technologies involved on your projects, little by little.
    a. https://logging.apache.org/log4j/2.x/
    b. https://www.slf4j.org/
    c. https://rest-assured.io/
    d. https://testng.org/doc/

e. Not all documentation is equally readable or comprehensive. Take the challenge to research outside of these links and spend a few minutes deepening your understanding on these technologies.

31. Right click on pom.xml and click on maven and then click “reload project”
32. Go to your src -> test -> java package in intellij and right click on it and add new Java class named VincentRestAssuredProjectTest.java, and add the following code to it. Let IntelliJ help you with the imports (don’t type those all out). Also, don’t add the intellisense support words because they’re not part of the code. Examples below are visible in light grey to the far right of some lines of code. You want to get used to discerning code versus intellisense built in to help you understand the meaning of parameters or the impact of methods or the results of methods.

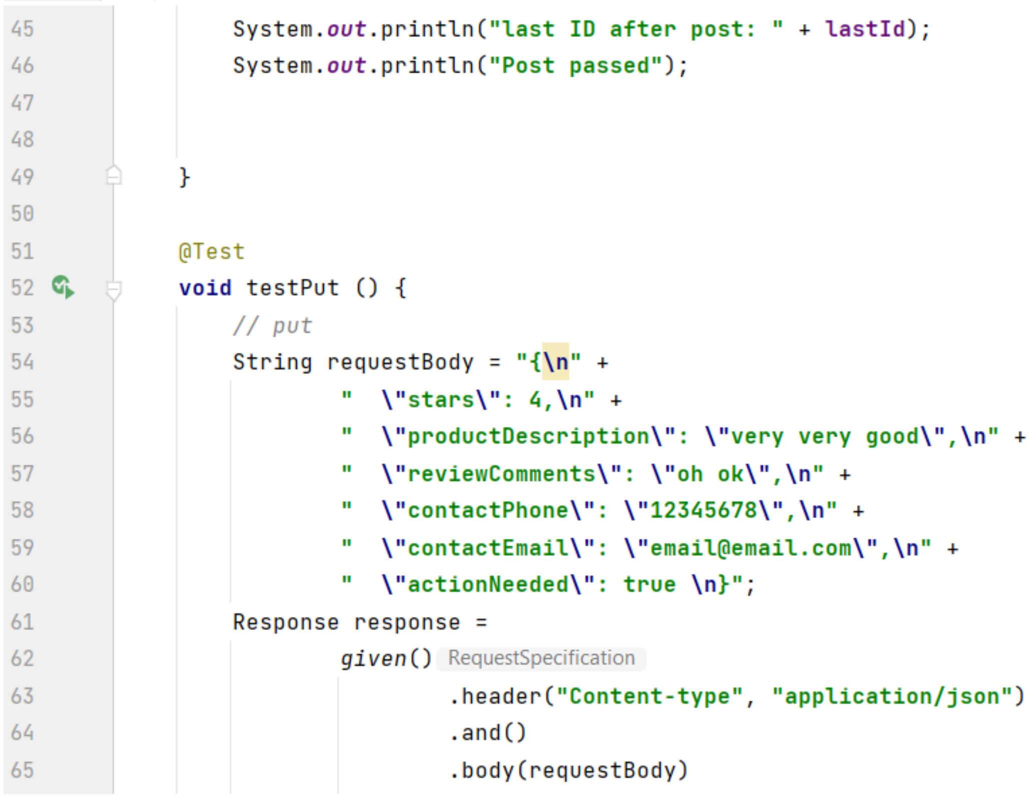
a. 

```java
import static io.restassured.RestAssured.*;
import io.restassured.response.Response;
import org.testng.Assert;
import org.testng.annotations.Test;

import static io.restassured.matcher.RestAssuredMatchers.*;
import static org.hamcrest.Matchers.*;
public class MongoDbRestApiTester {
    String port = "8090";
    String lastId = "";

    @Test
    void testGetAll () {
        Response response = get("http://localhost:" + port + "/api/reviews");
        System.out.println(response.asString());

        int statusCode = response.getStatusCode();

        //Assert
        Assert.assertEquals(statusCode, 204);
        System.out.println("Get all passed");
    }

```



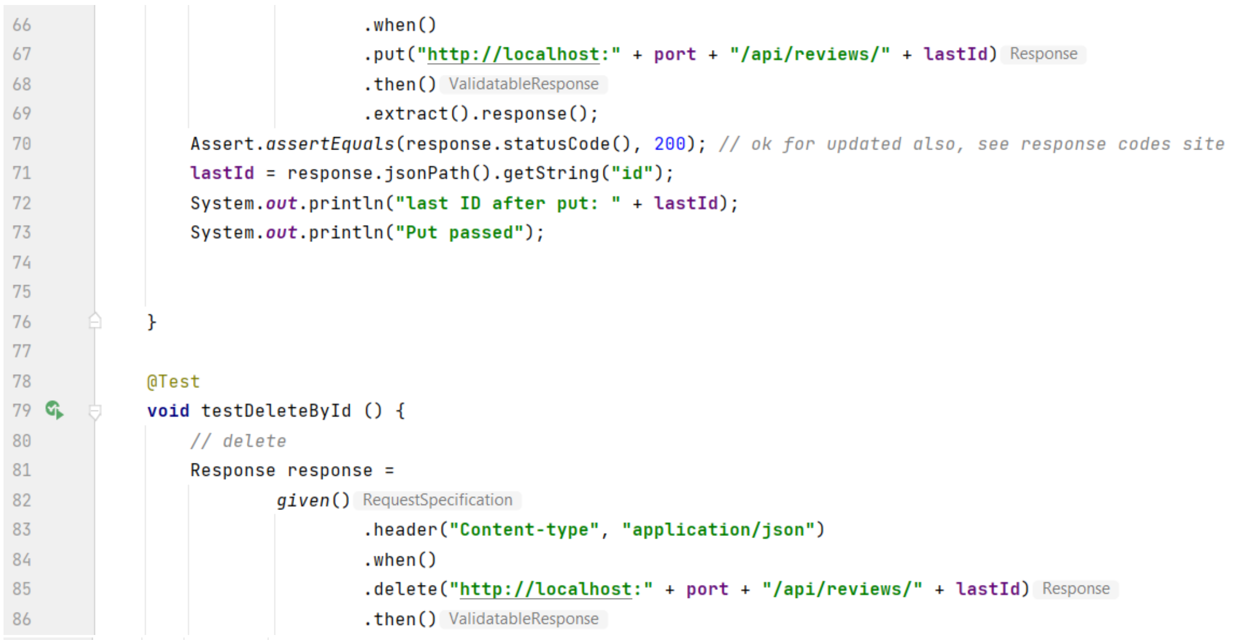
b.

```java
@Test
void testPost () {
    // post
    String requestBody = "{\n" +
            "  \"stars\": 3,\n" +
            "  \"productDescription\": \"very very good\",\n" +
            "  \"reviewComments\": \"oh ok\",\n" +
            "  \"contactPhone\": \"12345678\",\n" +
            "  \"contactEmail\": \"email@email.com\",\n" +
            "  \"actionNeeded\": false \n}";
    Response response =
            given() RequestSpecification
                    .header("Content-type", "application/json")
                    .and()
                    .body(requestBody)
                    .when()
                    .post("http://localhost:" + port + "/api/reviews/") Response
                    .then() ValidatableResponse
                    .extract().response();
    Assert.assertEquals(response.statusCode(), 201); // created, good, see response codes site
    lastId = response.jsonPath().getString("id");
```

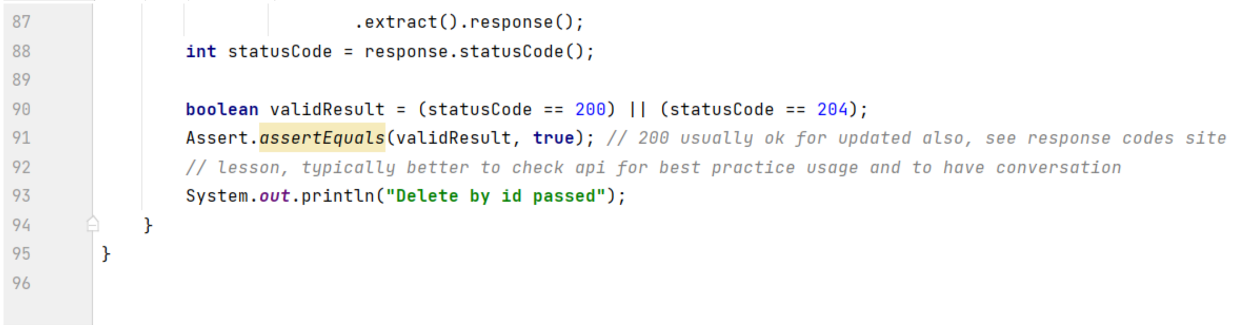


c.

```java
    System.out.println("last ID after post: " + lastId);
    System.out.println("Post passed");


}

@Test
void testPut () {
    // put
    String requestBody = "{\n" +
            "  \"stars\": 4,\n" +
            "  \"productDescription\": \"very very good\",\n" +
            "  \"reviewComments\": \"oh ok\",\n" +
            "  \"contactPhone\": \"12345678\",\n" +
            "  \"contactEmail\": \"email@email.com\",\n" +
            "  \"actionNeeded\": true \n}";
    Response response =
            given() RequestSpecification
                    .header("Content-type", "application/json")
                    .and()
                    .body(requestBody)
```

d. 

```
                .when()
                .put("http://localhost:" + port + "/api/reviews/" + lastId) Response
                .then() ValidatableResponse
                .extract().response();
        Assert.assertEquals(response.statusCode(), 200); // ok for updated also, see response codes site
        lastId = response.jsonPath().getString("id");
        System.out.println("last ID after put: " + lastId);
        System.out.println("Put passed");


    }

    @Test
    void testDeleteById () {
        // delete
        Response response =
                given() RequestSpecification
                        .header("Content-type", "application/json")
                        .when()
                        .delete("http://localhost:" + port + "/api/reviews/" + lastId) Response
                        .then() ValidatableResponse
```

e. 

```
                        .extract().response();
        int statusCode = response.statusCode();

        boolean validResult = (statusCode == 200) || (statusCode == 204);
        Assert.assertEquals(validResult, true); // 200 usually ok for updated also, see response codes site
        // lesson, typically better to check api for best practice usage and to have conversation
        System.out.println("Delete by id passed");
    }
}
```

33. Add testNg to classpath (using intellij help by right clicking on test annotation)
    a. Test annotation looks like this: @Test
34. Run the tests on the double arrow near VincentRestAssuredProjectTest so that all tests will run
35. If prompted, allow windows firewall access.
36. Wait to see the results. The results will give you feedback regarding 4 of your customer review api's via 4 test results.
37. Be aware of additional great references to continue your learning
    a. http://Automationstepbystep.com
    b. https://reqres.in/
38. Turn this part in by exporting your zip file from your intellij project and uploading your zip file into canvas (or for those outside of canvas, into github).
39. Celebrate! You're gaining more and more valuable experience that you can reflect upon, compare, ask questions about, and innovate upon.
40. (75 points) Extra Credit. But strongly recommended, and definitely excellent for your interview preparation.
    a. The total possible credit for this assignment including this extra credit is 175 points.
    b. Read the following 5 articles and include 20 things you learned (4 from each):
        i. Hamcrest Tutorial.
        ii. Assertions in TestNG | Selenium Easy
        iii. Behavior Driven Development - Gherkin (tutorialspoint.com)

iv. Spring Boot - Apache Kafka (tutorialspoint.com)
v. Secure REST API using Spring Boot | Toptal

c. Please also share any questions you have.
d. Turn this part in by uploading your notes and questions into canvas.

41. Great job! You're seeing more and more that won't be new to you when you become a professional, and more and more that will make sense to you.

Please Ask questions early and often.

Jvincent30@stlcc.edu

517-819-7805