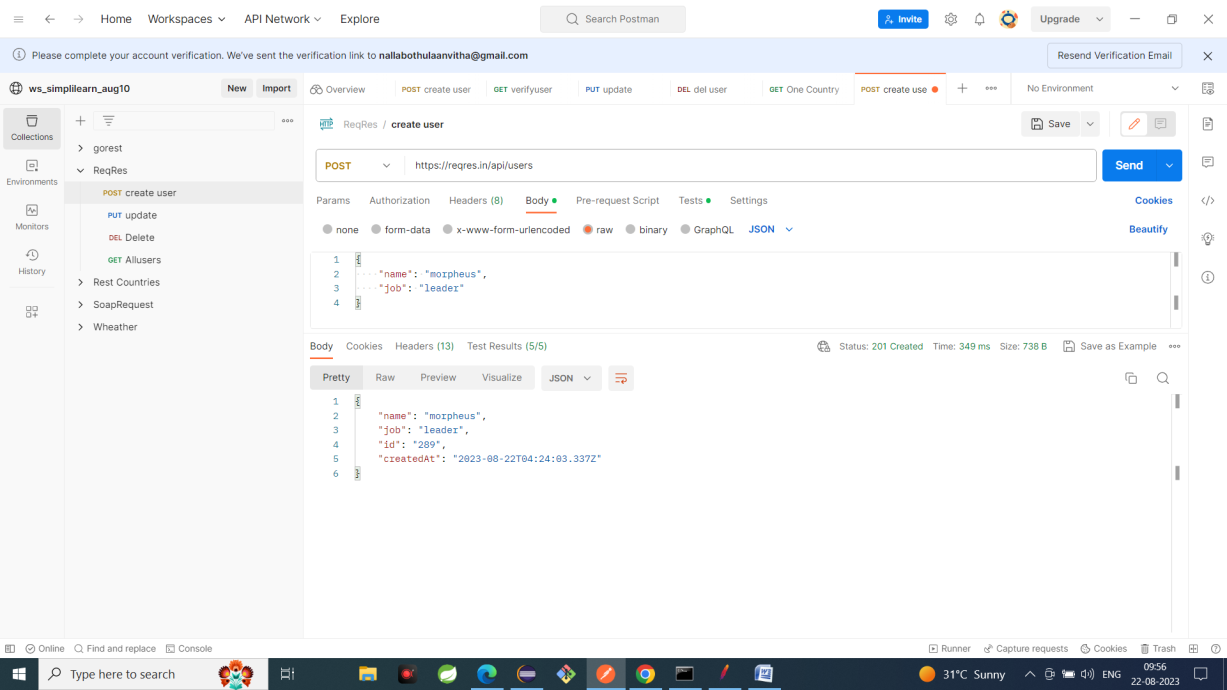
# SourceCode:

## Postman:

**PostRequest:**

I am sending this Url: <https://reqres.in/api/users> I got given response



**TestScriptsForResponse:**

pm.**test**("Response status code is 201", **function** () {

    pm.response.to.have.status(201);

});

pm.**test**("Response has the required fields", **function** () {

    const responseData **=** pm.response.json();

    pm.expect(responseData).to.be.an('object');

    pm.expect(responseData.name).to.exist.and.to.be.a('string');

    pm.expect(responseData.job).to.exist.and.to.be.a('string');

    pm.expect(responseData.id).to.exist.and.to.be.a('string');

    pm.expect(responseData.createdAt).to.exist.and.to.be.a('string');

});

pm.**test**("Name is a non-empty string", **function** () {

    const responseData **=** pm.response.json();

    pm.expect(responseData).to.be.an('object');

    pm.expect(responseData.name).to.be.a('string').and.to.have.lengthOf.at.least(1, "Value should not be empty");

});

pm.**test**("Job is a non-empty string", **function** () {

    const responseData **=** pm.response.json();

    pm.expect(responseData).to.be.an('object');

    pm.expect(responseData.job).to.be.a('string').and.to.have.lengthOf.at.least(1, "Job should not be empty");

});

pm.**test**("The id should be a non-empty string", **function** () {

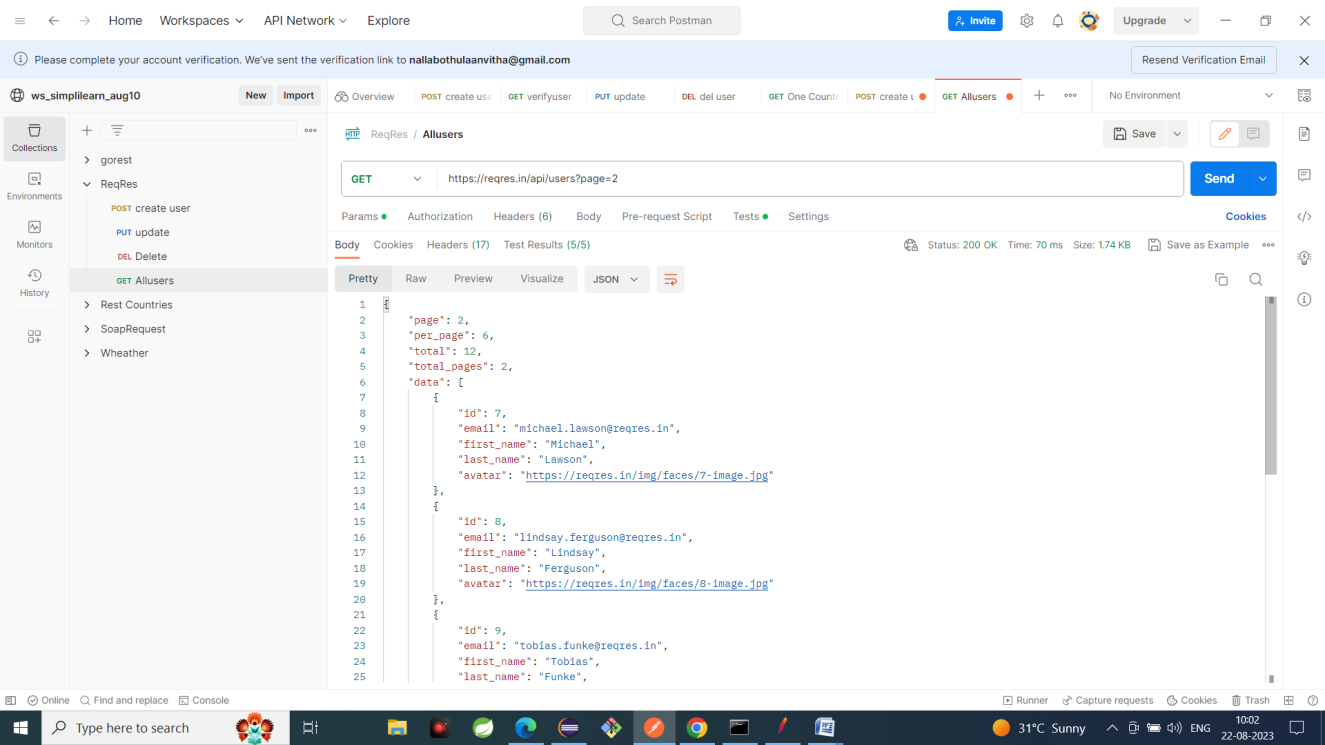
    const responseData **=** pm.response.json();

    pm.expect(responseData.id).to.be.a('string').and.to.have.lengthOf.at.least(1, "Value should not be empty");

});

**Getrequest:**

I am sending this Url: <https://reqres.in/api/users?page=2> I got given response



**TestScriptsForResponse:**

pm.**test**("Response status code is 200", **function** () {

    pm.expect(pm.response.code).to.equal(200);

});

pm.**test**("Validate the page field is a non-negative integer", **function** () {

    const responseData **=** pm.response.json();

    pm.expect(responseData.page).to.exist.and.to.be.a('number');

    pm.expect(responseData.page).to.be.at.least(0);

});

pm.**test**("Validate the 'per\_page' field is a non-negative integer", **function** () {

    const responseData **=** pm.response.json();

    pm.expect(responseData).to.be.an('object');

    pm.expect(responseData.per\_page).to.be.a('number').and.to.be.at.least(0, "'per\_page' should be a non-negative integer");

});

pm.**test**("Verify that the 'data' array is present and contains at least one element", **function** () {

    const responseData **=** pm.response.json();

    pm.expect(responseData).to.be.an('object');

    pm.expect(responseData.data).to.exist.and.to.be.an('array').with.lengthOf.at.least(1);

});

pm.**test**("Validate email format in data array", **function** () {

    const responseData **=** pm.response.json();

    pm.expect(responseData.data).to.be.an('array').and.to.have.lengthOf.at.least(1, "Data array should not be empty");

    responseData.data.**forEach**(**function** (user) {

        pm.expect(user.email).to.be.a('string').and.to.**match**(/**^**[\w-]**+**(\.[\w-]**+**)**\***@([\w-]**+**\.)**+**[a-zA-Z]**{2,7}$**/);

    });

});

# RestAssuredCode:

## GetPostCode:

package in.restassured.RestAssuredDemo;

import java.util.HashMap;

import java.util.UUID;

import org.junit.Assert;

import org.testng.annotations.BeforeTest;

import org.testng.annotations.Test;

import io.restassured.RestAssured;

import io.restassured.path.json.JsonPath;

import io.restassured.response.Response;

import static org.hamcrest.Matchers.is;

public class Get\_Post\_Demo extends TestBase {

HashMap<String, String> map = new HashMap<String, String>();

String id;

JsonPath jsonPath;

@Test(priority = 0)

public void payLoad() {

map.put("name", "morpheus");

map.put("job", "leader");

RestAssured.baseURI = "https://reqres.in/";

RestAssured.basePath = "/api/users";

logger.info("payload created");

}

@Test(priority = 1)

public void createResource() {

Response response = RestAssured.given().contentType("application/json").body(map)

.header("Authorization", "Bearer 49c69bcde74a7f84ec5a0020321939a6d643f0052a2d438a056760e5335ff446")

.when().post().then().extract().response();

logger.info("resource created sucessfully");

jsonPath = response.jsonPath();

id = jsonPath.get("id");

System.out.println("id");

}

@Test(priority = 2)

public void verifyResource() {

RestAssured.given().contentType("application/json")

.header("Authorization", "Bearer 49c69bcde74a7f84ec5a0020321939a6d643f0052a2d438a056760e5335ff446")

.when().get("https://reqres.in/api/users?"+id).then()

.statusCode(200);

logger.info("resource verified sucessfully");

Assert.assertTrue(jsonPath.get("name").toString().equals("morpheus"));

}

}

## TestBase:

package in.restassured.RestAssuredDemo;

import org.apache.log4j.Level;

import org.apache.log4j.Logger;

import org.apache.log4j.PropertyConfigurator;

import org.testng.annotations.BeforeClass;

import com.gargoylesoftware.htmlunit.javascript.host.fetch.Response;

import io.restassured.specification.RequestSpecification;

public class TestBase {

public static RequestSpecification httpRequest;

public static Response response;

public Logger logger;

@BeforeClass

public void setUp() {

logger = Logger.getLogger("LogDemo");

PropertyConfigurator.configure("log4j.properties");

logger.setLevel(Level.DEBUG);

}

}

## Log4j.propertiesfile:

log4j.rootLogger=ERROR,stdout

log4j.logger.com.endeca=INFO

log4j.logger.LogDemo=DEBUG, dest1

# Logger for crawl metrics

log4j.logger.com.endeca.itl.web.metrics=INFO

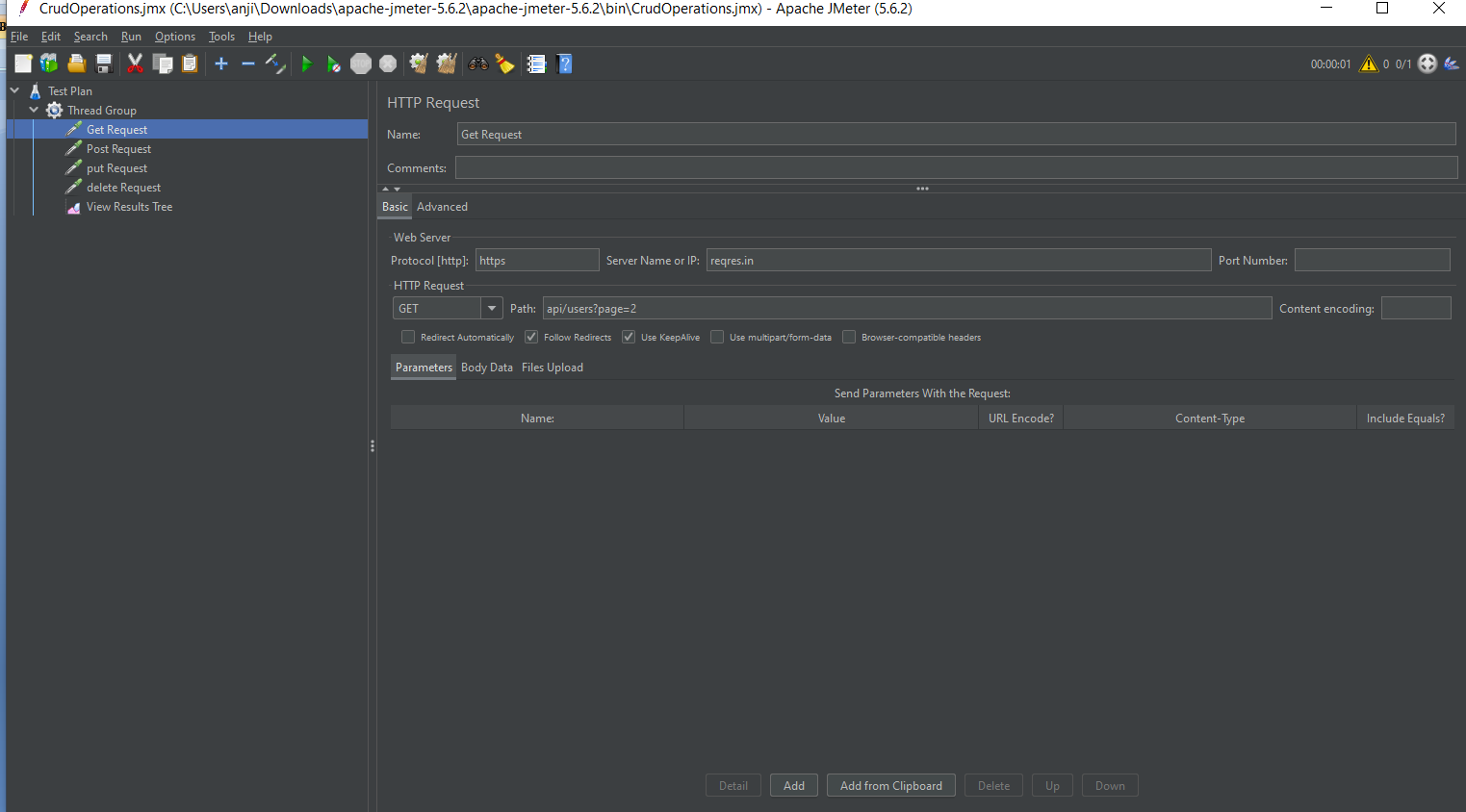
log4j.appender.stdout=org.apache.log4j.ConsoleAppender

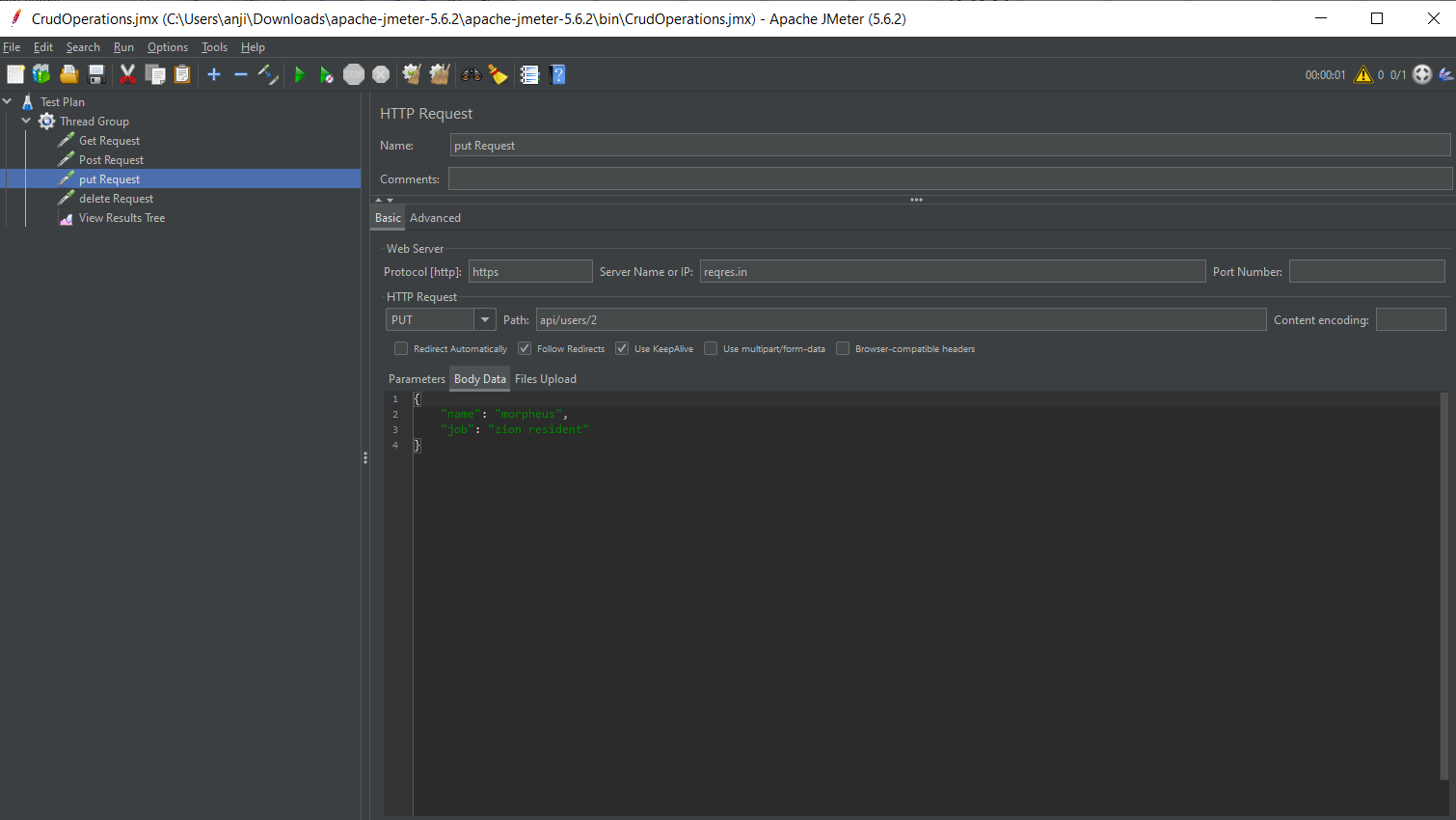
log4j.appender.stdout.layout=org.apache.log4j.PatternLayout

log4j.appender.stdout.layout.ConversionPattern=%p\t%d{ISO8601}\t%r\t%c\t[%t]\t%m%n

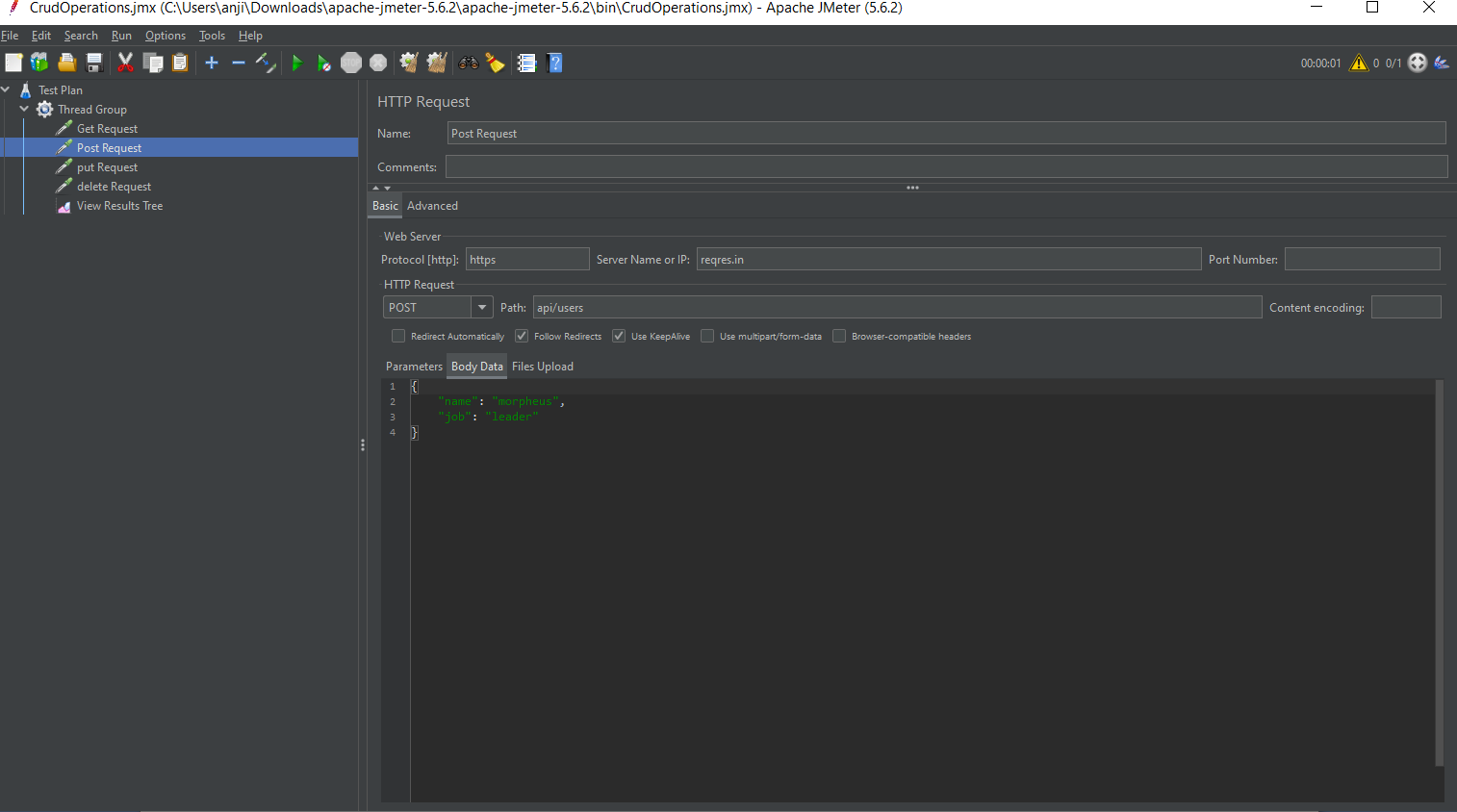
## JmeterCode:

**GetRequest:**

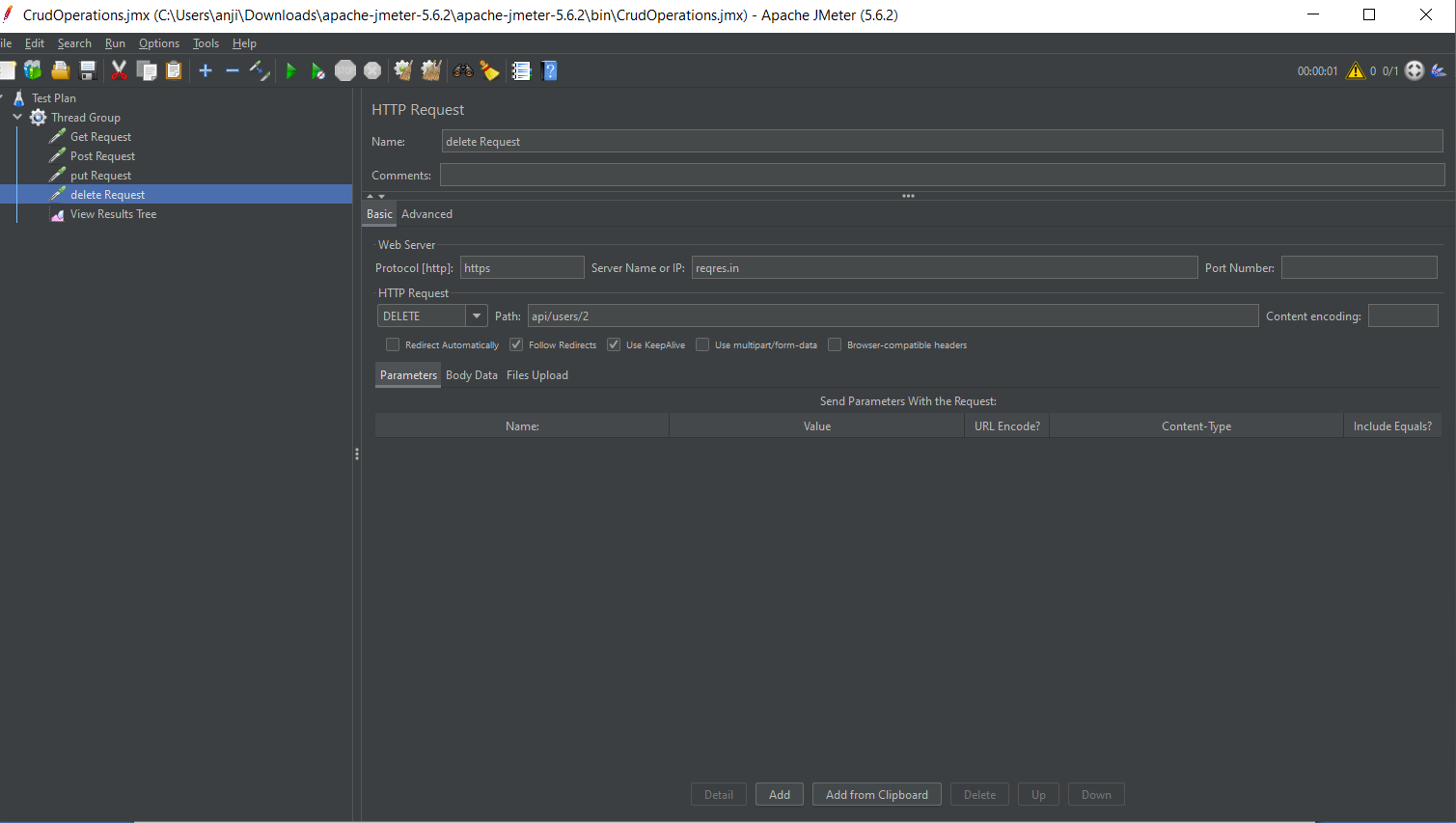
****

**putRequest:**

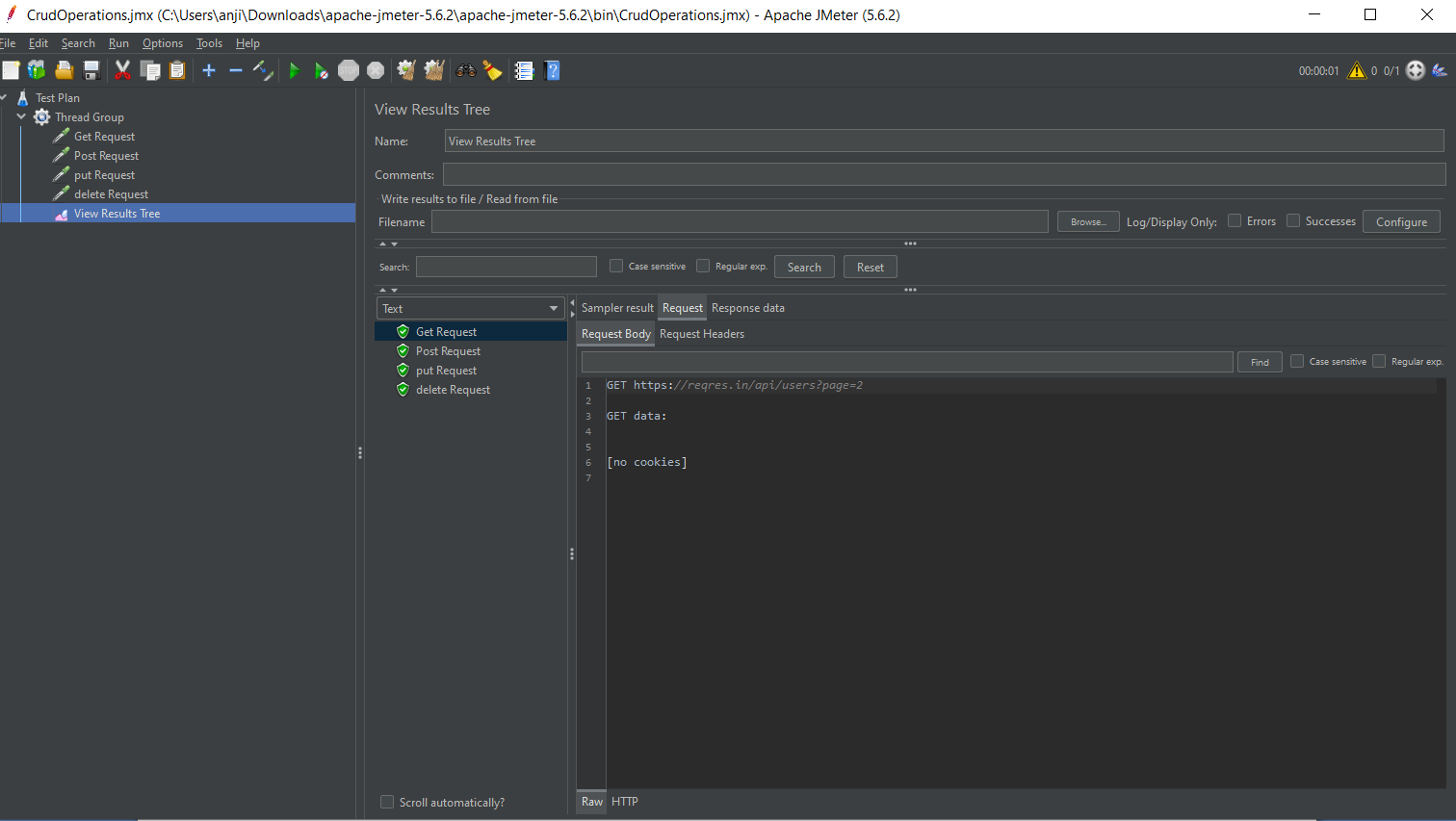
**PostRequest:**

****

**DeleteRequest:**

****

**ViewResultstree:**

****