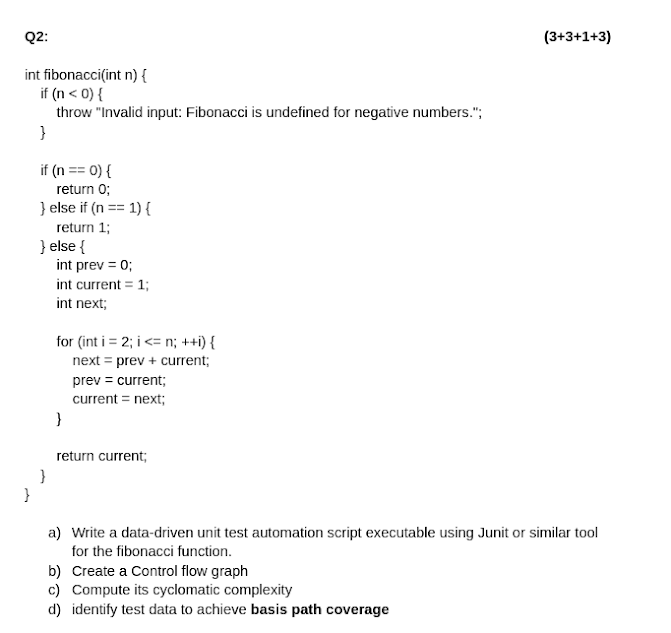
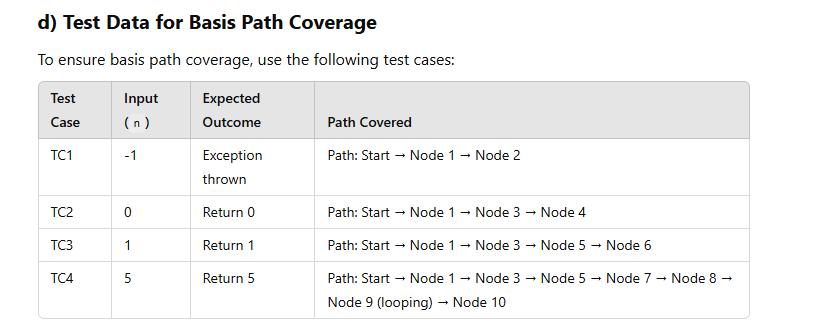
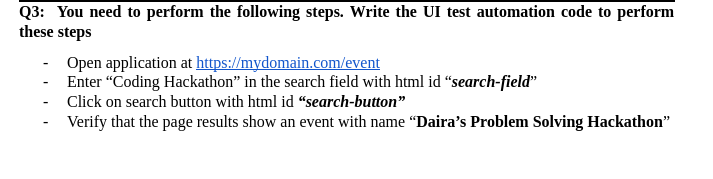
# 









**if (wait.until(ExpectedConditions.visibilityOfElementLocated(By.xpath("//\*[contains(text(), 'Daira’s Problem Solving Hackathon')]"))).isDisplayed())**



# UI Test Automation (Selenium + TestNG)

public class UITest {

WebDriver driver;

@BeforeClass

public void setUp() {

System.setProperty("webdriver.chrome.driver", "path/to/chromedriver");

driver = new ChromeDriver();

driver.get("https://example.com");

}

@Test

public void loginTest() {

WebElement username = driver.findElement(By.id("username"));

WebElement password = driver.findElement(By.id("password"));

WebElement loginButton = driver.findElement(By.id("login"));

username.sendKeys("validUser");

password.sendKeys("validPassword");

loginButton.click();

WebElement dashboard = driver.findElement(By.id("dashboard"));

Assert.assertTrue(dashboard.isDisplayed(), "Dashboard is not displayed after login");

}

@AfterClass

public void tearDown() {

if (driver != null) {

driver.quit();

}

}

}

# API Testing

public class APITest {

@BeforeClass

public void setUp() {

RestAssured.baseURI = "https://api.example.com";

}

@Test

public void getUserTest() {

Response response = RestAssured.given()

.header("Authorization", "Bearer your\_token\_here")

.get("/users/1");

Assert.assertEquals(response.getStatusCode(), 200);

Assert.assertEquals(response.jsonPath().getString("name"), "John Doe");

}

@Test

public void createUserTest() {

String payload = "{ \"name\": \"Jane Doe\", \"email\": \"jane@example.com\" }";

Response response = RestAssured.given()

.header("Content-Type", "application/json")

.body(payload)

.post("/users");

Assert.assertEquals(response.getStatusCode(), 201);

Assert.assertEquals(response.jsonPath().getString("name"), "Jane Doe");

}

}

# Complete Code with TestNG Annotations

import io.restassured.RestAssured;

import io.restassured.http.ContentType;

import io.restassured.response.Response;

import org.testng.Assert;

import org.testng.annotations.BeforeClass;

import org.testng.annotations.Test;

import static io.restassured.RestAssured.\*;

public class ApiTests {

@BeforeClass

public void setup() {

RestAssured.baseURI = "https://jsonplaceholder.typicode.com";

}

## // Scenario 1: Positive GET Request - Retrieve user details

@Test

public void testGetUserDetails\_Positive() {

Response response = given()

.when()

.get("/users/1")

.then()

.statusCode(200)

.extract()

.response();

// Validate response content

Assert.assertEquals(response.jsonPath().getString("name"), "Leanne Graham");

Assert.assertEquals(response.jsonPath().getString("username"), "Bret");

// Validate response time

Assert.assertTrue(response.getTime() < 2000, "Response time is too slow");

}

### // Scenario 2: Negative GET Request - Retrieve non-existing user

@Test

public void testGetUserDetails\_Negative() {

given()

.when()

.get("/users/9999")

.then()

.statusCode(404);

}

## // Scenario 3: Positive POST Request - Create a new user

@Test

public void testCreateUser\_Positive() {

String requestBody = """

{

"name": "John Doe",

"username": "johnd",

"email": "johndoe@example.com"

}

""";

Response response = given()

.contentType(ContentType.JSON)

.body(requestBody)

.when()

.post("/users")

.then()

.statusCode(201)

.extract()

.response();

// Validate created user details

Assert.assertEquals(response.jsonPath().getString("name"), "John Doe");

Assert.assertEquals(response.jsonPath().getString("username"), "johnd");

}

## // Scenario 4: Negative POST Request - Create a user with invalid data

@Test

public void testCreateUser\_Negative() {

String requestBody = """

{

"name": "", // Empty name

"username": "johnd",

"email": "invalid-email" // Invalid email format

}

""";

given()

.contentType(ContentType.JSON)

.body(requestBody)

.when()

.post("/users")

.then()

.statusCode(400); // Expecting 400 Bad Request

}

## // Scenario 5: PUT Request - Update an existing user's information

@Test

public void testUpdateUser\_Positive() {

String requestBody = """

{

"name": "Jane Doe",

"username": "janed",

"email": "janedoe@example.com"

}

""";

Response response = given()

.contentType(ContentType.JSON)

.body(requestBody)

.when()

.put("/users/1")

.then()

.statusCode(200)

.extract()

.response();

// Verify updated data

Assert.assertEquals(response.jsonPath().getString("name"), "Jane Doe");

Assert.assertEquals(response.jsonPath().getString("username"), "janed");

}

## // Scenario 6: DELETE Request - Delete an existing user

@Test

public void testDeleteUser\_Positive() {

given()

.when()

.delete("/users/1")

.then()

.statusCode(200);

// Verify deletion by trying to get the user again

given()

.when()

.get("/users/1")

.then()

.statusCode(404);

}

## // Scenario 7: Check response headers for GET request

@Test

public void testGetUserHeaders() {

Response response = given()

.when()

.get("/users/1")

.then()

.statusCode(200)

.extract()

.response();

// Verify content type

Assert.assertEquals(response.getHeader("Content-Type"), "application/json; charset=utf-8");

}

## // Scenario 8: Validate response structure for important fields

@Test

public void testResponseStructureValidation() {

given()

.when()

.get("/users/1")

.then()

.statusCode(200)

.body("id", org.hamcrest.Matchers.equalTo(1))

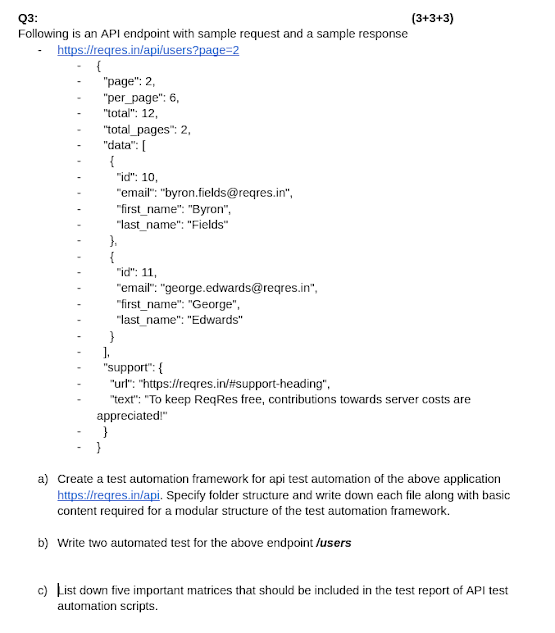
.body("name", org.hamcrest.Matchers.notNullValue())

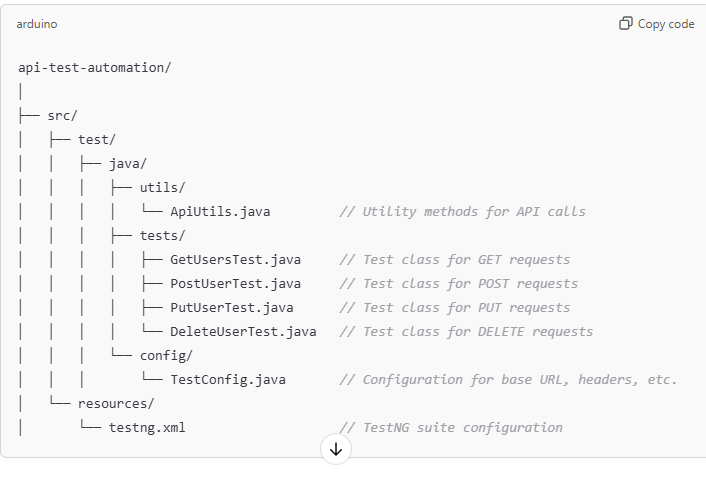
.body("username", org.hamcrest.Matchers.notNullValue())

.body("email", org.hamcrest.Matchers.notNullValue());

}

}





b) Automated Tests for /users Endpoint

import io.restassured.RestAssured;

import io.restassured.http.ContentType;

import io.restassured.response.Response;

import org.testng.Assert;

import org.testng.annotations.BeforeClass;

import org.testng.annotations.Test;

public class UserApiTests {

@BeforeClass

public void setup() {

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

}

// GET request to fetch user details

@Test

public void testGetUsers() {

Response response = RestAssured.get("/users?page=2");

Assert.assertEquals(response.getStatusCode(), 200);

Assert.assertTrue(response.jsonPath().getList("data").size() > 0, "User list is empty");

}

// POST request to create a new user

@Test

public void testCreateUser() {

String requestBody = "{\"name\": \"morpheus\", \"job\": \"leader\"}";

Response response = RestAssured.given()

.contentType(ContentType.JSON)

.body(requestBody)

.post("/users");

Assert.assertEquals(response.getStatusCode(), 201);

Assert.assertEquals(response.jsonPath().getString("name"), "morpheus");

}

// PUT request to update an existing user

@Test

public void testUpdateUser() {

String requestBody = "{\"name\": \"morpheus\", \"job\": \"zion resident\"}";

Response response = RestAssured.given()

.contentType(ContentType.JSON)

.body(requestBody)

.put("/users/2");

Assert.assertEquals(response.getStatusCode(), 200);

Assert.assertEquals(response.jsonPath().getString("job"), "zion resident");

}

// DELETE request to delete a user

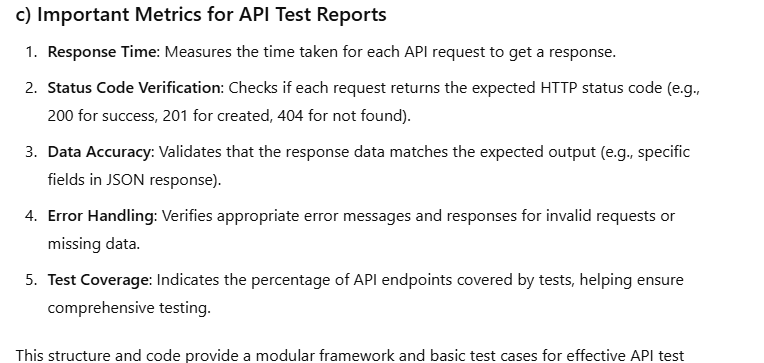
@Test

public void testDeleteUser() {

Response response = RestAssured.delete("/users/2");

Assert.assertEquals(response.getStatusCode(), 204);

}

} 

# **UNIT Testing**

package sqe\_CP;

import org.testng.annotations.DataProvider;

import org.testng.annotations.Test;

import org.testng.asserts.SoftAssert;

public class SimpleCalculator {

// Calculator methods

public int add(int a, int b) {

return a + b;

}

public int sub(int a, int b) {

return a - b;

}

public int multiply(int a, int b) {

return a \* b;

}

public int divide(int a, int b) {

if (b == 0) {

throw new IllegalArgumentException("Division by zero is not allowed.");

}

return a / b;

}

// Test methods

@Test(priority = 1, dataProvider = "addData")

public void testAdd(int a, int b, int expected) {

SoftAssert softAssert = new SoftAssert();

int actual = add(a, b);

softAssert.assertEquals(actual, expected, "Addition result is incorrect");

softAssert.assertAll();

}

@Test(priority = 2, dataProvider = "subData")

public void testSub(int a, int b, int expected) {

SoftAssert softAssert = new SoftAssert();

int actual = sub(a, b);

softAssert.assertEquals(actual, expected, "Subtraction result is incorrect");

softAssert.assertAll();

}

@Test(priority = 3, dataProvider = "multiplyData")

public void testMultiply(int a, int b, int expected) {

SoftAssert softAssert = new SoftAssert();

int actual = multiply(a, b);

softAssert.assertEquals(actual, expected, "Multiplication result is incorrect");

softAssert.assertAll();

}

@Test(priority = 4, dataProvider = "divideData")

public void testDivide(int a, int b, int expected) {

SoftAssert softAssert = new SoftAssert();

int actual = divide(a, b);

softAssert.assertEquals(actual, expected, "Division result is incorrect");

softAssert.assertAll();

}

@Test(priority = 5, expectedExceptions = IllegalArgumentException.class)

public void testDivideByZero() {

divide(10, 0); // Only check for exception

}

// Data Providers

@DataProvider(name = "addData")

public Object[][] getAddData() {

return new Object[][] {

{2, 3, 5},

{10, 5, 15},

{0, 0, 0}

};

}

@DataProvider(name = "subData")

public Object[][] getSubData() {

return new Object[][] {

{5, 3, 2},

{10, 5, 5},

{0, 0, 0}

};

}

@DataProvider(name = "multiplyData")

public Object[][] getMultiplyData() {

return new Object[][] {

{2, 3, 6},

{10, 5, 50},

{0, 5, 0}

};

}

@DataProvider(name = "divideData")

public Object[][] getDivideData() {

return new Object[][] {

{6, 3, 2},

{10, 2, 5},

{9, 3, 3}

};

}

}