

Zephyr Integration (Zapi)

Ostan Dsouza
CodeCraft Technologies
10/06/2020

Table of Contents

Chapter 1: Introduction	3
Section 1.1: Features of Zephyr for JIRA	3
Chapter 2: Zephyr Api(ZAPI)	3
Section 2.1: Advantages of Zapi	3
Section 2.2: Prerequisites	4
Chapter 3: ZAPI Integration	4
Section 3.1: API Keys	4
Section 3.2: Creating APIs	6
Section 3.3: JWT Token	11
Chapter 4: Framework Changes	12
Section 4.1: Zephyr Logic	12

Chapter 1: Introduction

Zephyr provides a suite of tools to optimize speed and quality of software testing, empowering you with the flexibility, visibility, and insights you need to achieve Continuous Testing Agility. Zephyr for JIRA is a native application that exists in JIRA and brings quality test management capabilities to any JIRA project. When Zephyr is used with JIRA, the test can be created, viewed in any JIRA project, and executed immediately or as part of a testing cycle that may be linked to other issues

Section 1.1: Features of Zephyr for JIRA

- Native to JIRA allowing users to test right inside JIRA
 - Testing is integrated into the project cycles and it enables you to track software quality and make empowered go/no-go decisions.
- Create, Plan, and Execute Tests
 - Create, view, and modify test steps and attachments for individual tests. Build test execution cycles, execute the tests, and link defects to specific tests.
- Track Quality Metrics
 - Zephyr provides easy-to-use dashboard which provides testing metrics on the testing activities throughout every project.

Chapter 2: Zephyr Api(ZAPI)

ZAPI exposes Zephyr for JIRA data via REST APIs which allows you to build your own integration with Zephyr for JIRA. ZAPI (or Zephyr API) is an Add-on to Zephyr for JIRA that allows access to its testing data including the ability to view and upload information programmatically.

Section 2.1: Advantages of Zapi

Using ZAPI's capabilities, Zephyr for JIRA users can:

- Integrate with test automation tools
- Integrate with continuous integration tools
- Create extensive custom reports for testing
- Integrate with business intelligence tools
- Use the testing data for other purposes

Section 2.2: Prerequisites

There are a few quick things to be aware of about ZAPI:

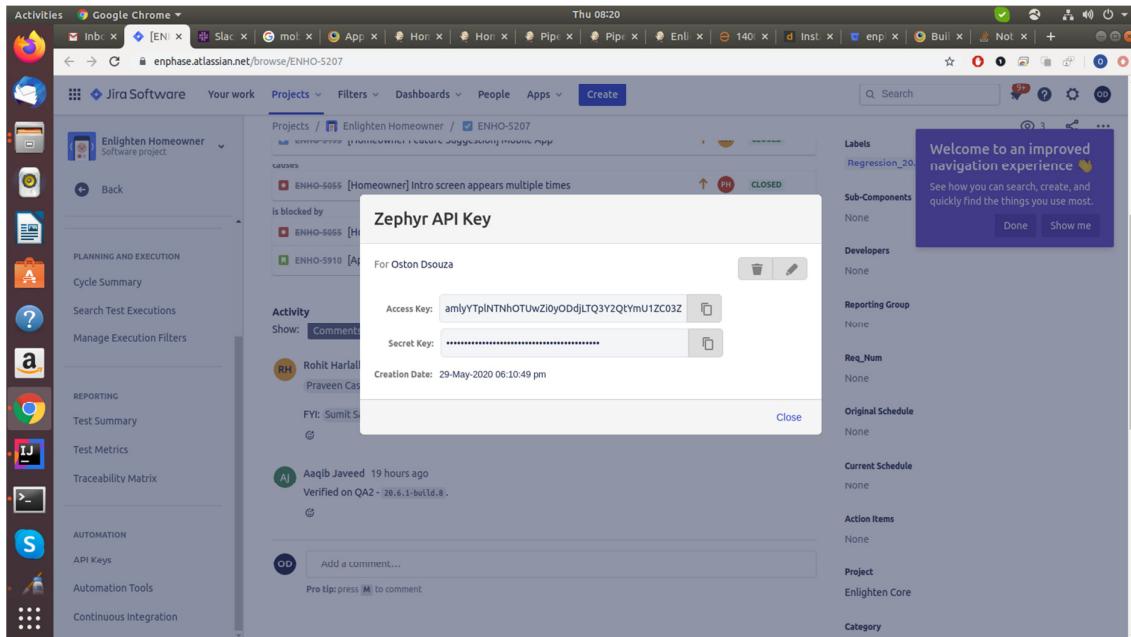
- ZAPI works only with Zephyr for JIRA Server versions 2 and higher
- A valid license of Zephyr for JIRA Server is needed for ZAPI to work
- ZAPI is only available for Zephyr for JIRA Server
- The API is entirely HTTP-based
- Parameters have certain expectations
- There are pagination limits
- Encoding affects status character counts
- No anonymous access is allowed; the security model is the same as that of Zephyr for JIRA (which in turn uses JIRA's security model)

Chapter 3: ZAPI Integration

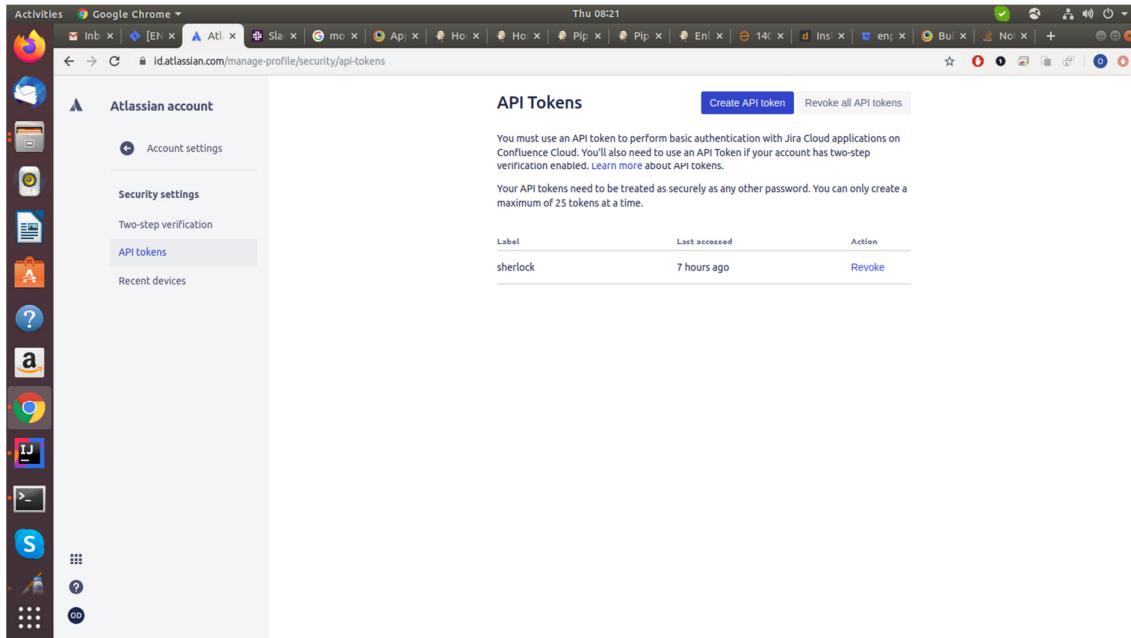
Follow Below Steps for integration zephyr with automation framework.

Section 3.1: API Keys

Once After Zapi is installed, you may get api keys for zapi simply by navigating to Zephyr->Automation->API Keys

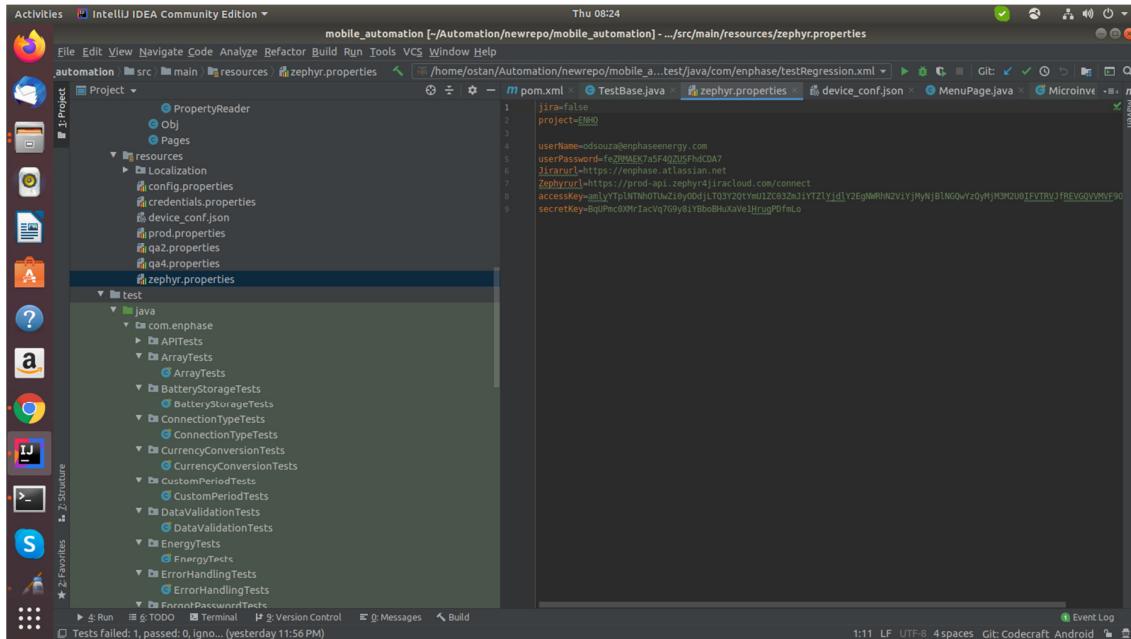


For Jira Authentication, you may generate API token simply by navigating to Accounts settings->Security->Create and manage API Token-> generate API Token



Zephyr Property:

Create Zephyr.property file inside root directory containing user and api tokens required for authentications along with jira flag that can enable or disable this feature.



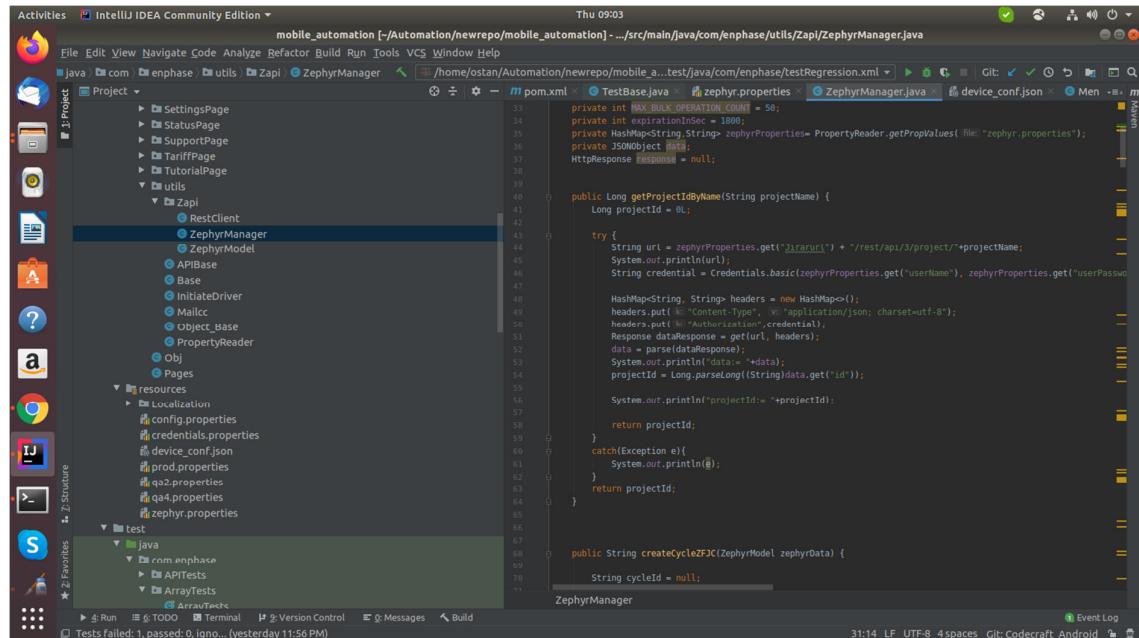
Section 3.2: Creating APIs

a) Get Project ID:

This is REST call to Jira API to get required project ID for specified project name. API document for this can be found [here](#).

GET /rest/api/3/project/{projectIdOrKey}

projectIdOrKey	String	REQUIRED
----------------	--------	----------



```

private static final int MAXIMUM_WAIT = 50;
private int expirationInSec = 1800;
private HashMap<String> zephyrProperties= PropertyReader.getPropValues( file: "zephyr.properties");
private JSONObject base;
HttpServletResponse response = null;

public Long getProjectIdByName(String projectName) {
    Long projectId = 0L;
    try {
        String url = zephyrProperties.get("zephyrurl") + "/rest/api/3/project/" + projectName;
        System.out.println(url);
        String credential = Credentials.basic(zephyrProperties.get("userName"), zephyrProperties.get("userPasswo
        HashMap<String> String-headers = new HashMap<String>();
        headers.put( "Content-Type", "application/json; charset=utf-8");
        headers.put( "Authorization", credential);
        Response dataResponse = get(url, headers);
        data = parse(dataResponse);
        System.out.println("data=" + data);
        projectId = Long.parseLong((String) data.get("id"));

        System.out.println("projectId=" + projectId);
    } catch(Exception e){
        System.out.println(e);
    }
    return projectId;
}

public String createCycleZFJC(ZephyrModel zephyrData) {
    String cycleId = null;
}

```

b) Create Cycle:

This is a REST call to zapi to create cycle on zephyr for specified project ID. API document for this can be found [here](#).

<https://prod-api.zephyr4jiracloud.com/connect/public/rest/api/1.0/cycle?expand=&clonedCycleId=>

name	String	REQUIRED
build	String	OPTIONAL
environment	String	OPTIONAL
description	String	OPTIONAL
startDate	String	OPTIONAL
endDate	String	OPTIONAL
projectId	String	REQUIRED
versionId	String	REQUIRED

```

    public String createCycleFC(ZephyrModel zephyrData) {
        String cycleId = null;
        try {
            String createCycleURL = zephyrProperties.get("zephyrurl") + "/public/rest/api/1.0/cycle";
            System.out.println("createCycleURL=" + createCycleURL);
            ZFCloudRestClient client = ZFCloudRestClient.restBuilder(zephyrProperties.get("zephyrurl"), zephyrProperties.get("zephyrkey"));
            JwtGenerator jwtGenerator = client.getJwtGenerator();
            Uri uri = new Uri(createCycleURL);
            String jwtHeaderValue = jwtGenerator.generateJWT("POST", uri, expirationInSec);
            Date date = new Date();
            SimpleDateFormat sdf = new SimpleDateFormat("E dd, yyyy hh:mm a");
            String dateFormatForCycleCreation = sdf.format(date);
            String cycleName = zephyrData.getCyclePrefix() + dateFormatForCycleCreation;
            String cycleName = zephyrData.getCyclePrefix();
            SimpleDateFormat sdf1 = new SimpleDateFormat("yyyy-MM-dd");
            String startDate = sdf1.format(date);
            GregorianCalendar gCal = new GregorianCalendar();
            if (zephyrData.getCycleDuration().trim().equalsIgnoreCase("30 days")) {
                gCal.add(Calendar.DAY_OF_MONTH, -29);
            } else if (zephyrData.getCycleDuration().trim().equalsIgnoreCase("7 days")) {
                gCal.add(Calendar.DAY_OF_MONTH, -6);
            } else if (zephyrData.getCycleDuration().trim().equalsIgnoreCase("1 day")) {
                gCal.add(Calendar.DAY_OF_MONTH, -1);
            }
            String endDate = sdf1.format(gCal.getTime());
            HttpResponse response = null;
        }
    }

```

c) Add TestCases:

This is a REST call to zapi to add testcases on zephyr for specified cycle ID.
API document for this can be found [here](#)

<https://prod-api.zephyr4jiracloud.com/connect/public/rest/api/1.0/executions/add/cycle/cycleId>

cycleId	String	REQUIRED
---------	--------	-----------------

The screenshot shows the IntelliJ IDEA Community Edition interface. The top menu bar includes File, Edit, View, Navigate, Code, Analyze, Refactor, Build, Run, Tools, VCS, Window, and Help. The title bar indicates the project is "mobile_automation [-/Automation/newrepo/mobile_automation]" and the file being edited is "ZephyrManager.java". The status bar at the bottom shows the time as "Thu 09:03" and the date as "31:14 LF UTF-8 4 spaces Git: Codecraft_Android".

The left sidebar displays the project structure under "1-Project". The "mobile_automation" package contains several sub-packages and files:

- Java: com, enphase, utils, Zap!, ZephyrManager
- Project: pom.xml, TestBase.java, zephyr.properties, ZephyrManager.java, device_conf.json
- 1-Project: SettingsPage, StatusPage, SupportPage, TariffPage, TutorialPage, utils (containing Zapi, RestClient, ZephyrManager, ZephyrModel, APIBase, Base, InitiateDriver, Mailcc, Object_Base, PropertyReader, Obj, Pages), resources (Localization, config.properties, credentials.properties, device_conf.json, prod.properties, qa2.properties, qa4.properties, zephyr.properties), test (com.enphase, APITests, ArrayTests, ArrayTests)

The right pane shows the code for the `ZephyrManager` class. The specific method highlighted is `assignTestsToCycle()`:

```
public void assignTestsToCycle(ZephyrModel zephyrData, HashMap<String, Object> jsonObject) {
    try {
        String assignTestsToCycleURL = zephyrProperties.get("zephyrurl") + "/public/rest/api/1.0/executions/assignTestsToCycle";
        System.out.println("assignTestsToCycleURL=" + assignTestsToCycleURL);
        System.out.println("jsonObject=" + jsonObject.toString());
        ZFJCloudRestClient client = ZFJCloudRestClient.restBuilder(zephyrProperties.get("zephyrurl"), zephyrProperties.get("zephyrAccessKey"));
        JwtGenerator jwtGenerator = client.getJwtGenerator();
        URI uri = new URI(assignTestsToCycleURL);
        String jwtHeaderValue = jwtGenerator.generateJWT("POST", uri, expirationInSec);

        StringEntity se = new StringEntity(jsonObject.toString());

        HttpPost createCycleRequest = new HttpPost(assignTestsToCycleURL);

        createCycleRequest.addHeader("Content-Type", "application/json");
        createCycleRequest.addHeader("Authorization", jwtHeaderValue);
        createCycleRequest.addHeader("zapiAccessKey", zephyrProperties.get("accessKey"));

        createCycleRequest.setEntity(se);
        response = RestClient.getHttpClient().execute(createCycleRequest, RestClient.getContext(zephyrProperties));
        int statusCode = response.getStatusLine().getStatusCode();
        System.out.println("statusCode=" + statusCode);
        String token = null;
        if (statusCode >= 200 && statusCode < 300) {
            HttpEntity entity = response.getEntity();
            token = EntityUtils.toString(entity);
            System.out.println("string:" + token);
        }

        int maxTryCount = 0;
        boolean checkJobProgress = false;

        while (!checkJobProgress) {
            maxTryCount++;
            try {
                Thread.sleep(1000);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
            checkJobProgress = checkJobProgress();
        }
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

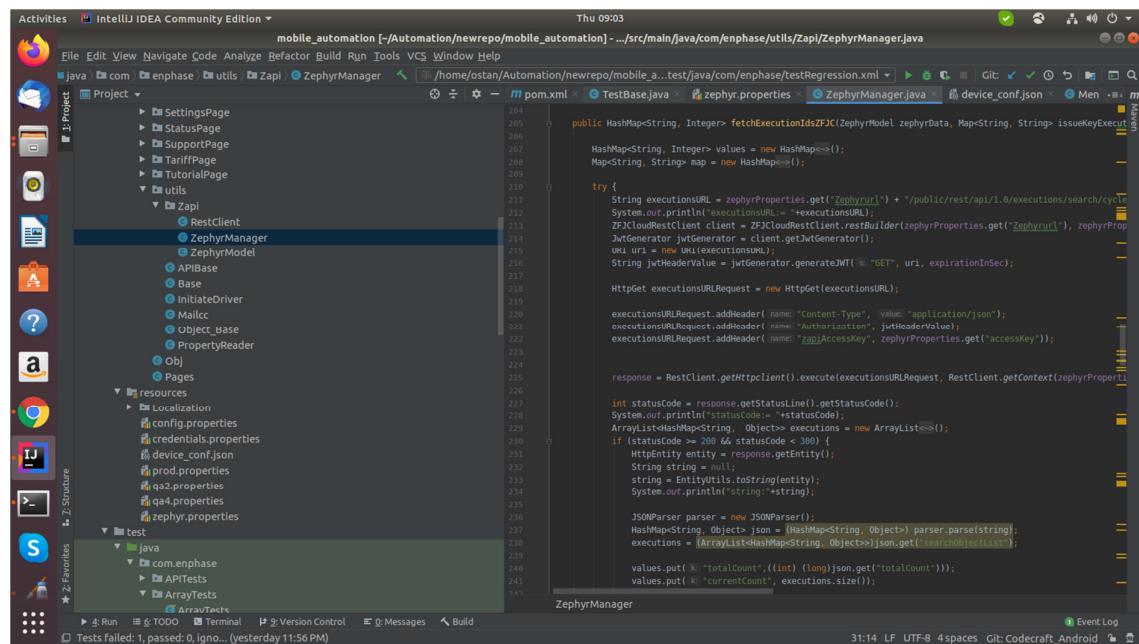
The status bar at the bottom also shows "Tests failed: 1, passed: 0, ignored: 0 (yesterday 11:56 PM)".

d) Fetch ExecutionID's:

This is a REST call to zapi to fetch execution ID's for all testcases added to Cycle. API document for this can be found [here](#)

<https://prod-api.zephyr4jiracloud.com/connect/public/rest/api/1.0/executions/search/cycle/cycleId?versionId=&expand=&offset=&size=&sortOrder=&action=&sortBy=&projectId=>

cycleId	String	REQUIRED
projectId	String	REQUIRED
versionId	String	REQUIRED



```

public HashMap<String, Integer> fetchExecutionIdsZFC(ZephyrModel zephyrData, Map<String, String> issueKeyExecutionMap) {
    HashMap<String, Integer> values = new HashMap<>();
    Map<String, String> map = new HashMap<>();

    try {
        String executionsURL = zephyrProperties.get("zephyrurl") + "/public/rest/api/1.0/executions/search/cycle";
        System.out.println("executionsURL=" + executionsURL);
        ZFCloudRestClient client = ZFCloudRestClient.restBuilder(zephyrProperties.get("zephyrurl"), zephyrProperties.get("zephyrAccessKey"), jwtGenerator = JwtGenerator.getJwtGenerator());
        URL url = new URL(executionsURL);
        String jwtHeaderValue = jwtGenerator.generateJWT("GET", url, expirationInSec);

        HttpGet executionsURLRequest = new HttpGet(executionsURL);
        executionsURLRequest.addHeader("Content-Type", "application/json");
        executionsURLRequest.addHeader("Authorization", jwtHeaderValue);
        executionsURLRequest.addHeader("ZAPIAccessKey", zephyrProperties.get("accessKey"));

        response = RestClient.getHttpClient().execute(executionsURLRequest, RestClient.getContext(zephyrProperties));
        int statusCode = response.getStatusLine().getStatusCode();
        System.out.println("statusCode=" + statusCode);
        ArrayList<HashMap<String, Object>> executions = new ArrayList<>();
        if (statusCode >= 200 && statusCode < 300) {
            HttpEntity entity = response.getEntity();
            String string = null;
            string = EntityUtils.toString(entity);
            System.out.println("string=" + string);

            JSONParser parser = new JSONParser();
            HashMap<String, Object> json = (HashMap<String, Object>) parser.parse(string);
            executions = (ArrayList<HashMap<String, Object>>) json.get("searchObjectList");

            values.put("totalCount", ((int) (long) json.get("totalCount")));
            values.put("currentCount", executions.size());
        }
    } catch (Exception e) {
        e.printStackTrace();
    }
}

```

e) Execute Tests:

This is a REST call to zapi to execute test based on execution Id's. API document for this can be found [here](#)

<https://prod-api.zephyr4jiracloud.com/connect/public/rest/api/1.0/executions>

cycleId	String	REQUIRED
---------	--------	----------

```

    public void executeTestsZFC(ZephyrModel zephyrData, List<String> passList, List<String> failList) {
        try {
            String bulkExecuteTestsURL = zephyrProperties.get("zephyrurl") + "/public/rest/api/1.0/executions";
            System.out.println("bulkExecuteTestsURL: " + bulkExecuteTestsURL);
            ZFCloudRestClient client = ZFCloudRestClient.restBuilder(zephyrProperties.get("zephyrurl"), zephyrProperties);
            JwtGenerator jwtGenerator = client.getJwtGenerator();
            Uri uri = new Uri(bulkExecuteTestsURL);
            String jwtHeaderValue = jwtGenerator.generateJWT("POST", uri, expirationInSec);

            int failListSize = failList.size();
            if (failListSize > 0) {
                int bulkOperationSetCount = failListSize / MAX_BULK_OPERATION_COUNT;
                bulkOperationSetCount += (failListSize % MAX_BULK_OPERATION_COUNT) > 0 ? 1 : 0;

                for (int i = 0; i < bulkOperationSetCount; i++) {
                    List<String> tempFailList = failList.subList((i * MAX_BULK_OPERATION_COUNT), ((i + 1) * MAX_BULK_OPERATION_COUNT));
                    JSONArray failedTests = new JSONArray();

                    for (String failedTest : tempFailList) {
                        failedTests.add(failedTest);
                    }

                    JSONObject failObj = new JSONObject();
                    failObj.put("executions", failedTests);
                    failObj.put("status", "2");
                    failObj.put("stepStatus", "1");
                    failObj.put("testStepStatusChangeFlag", true);
                    failObj.put("clearDefectMappingFlag", false);
                    StringEntity failEntity = new StringEntity(failObj.toString());

                    HttpPost bulkUpdateFailedTests = new HttpPost(bulkExecuteTestsURL);
                    bulkUpdateFailedTests.setHeader("Content-Type", "application/json");
                    bulkUpdateFailedTests.setHeader("Authorization", jwtHeaderValue);
                    bulkUpdateFailedTests.setHeader("zapiAccessKey", zephyrProperties.get("accessKey"));
                    bulkUpdateFailedTests.setEntity(failEntity);
                }
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}

```

f) Check Progress:

This is a REST call to zapi to get the update on job progress based on job progress token when bulk update is done. API document for this can be found [here](#)

<https://prod-api.zephyr4jiracloud.com/connect/public/rest/api/1.0/jobprogress/jobProgressTicket>

jobProgressTicket String **REQUIRED**

```

    private boolean checkJobProgress(ZephyrModel zephyrData, String token) {
        boolean jobCompleted = false;
        try {
            TimeUnit.SECONDS.sleep(1);
            String url = zephyrProperties.get("zephyrurl") + "/public/rest/api/1.0/jobprogress/" + token;
            System.out.println("url: " + url);
            ZFCloudRestClient client = ZFCloudRestClient.restBuilder(zephyrProperties.get("zephyrurl"), zephyrProperties);
            JwtGenerator jwtGenerator = client.getJwtGenerator();
            Uri uri = new Uri(url);
            String jwtHeaderValue = jwtGenerator.generateJWT("GET", uri, expirationInSec);

            HttpGet jobProgressRequest = new HttpGet(uri);

            jobProgressRequest.setHeader("Content-Type", "application/json");
            jobProgressRequest.setHeader("Authorization", jwtHeaderValue);
            jobProgressRequest.setHeader("zapiAccessKey", zephyrProperties.get("accessKey"));

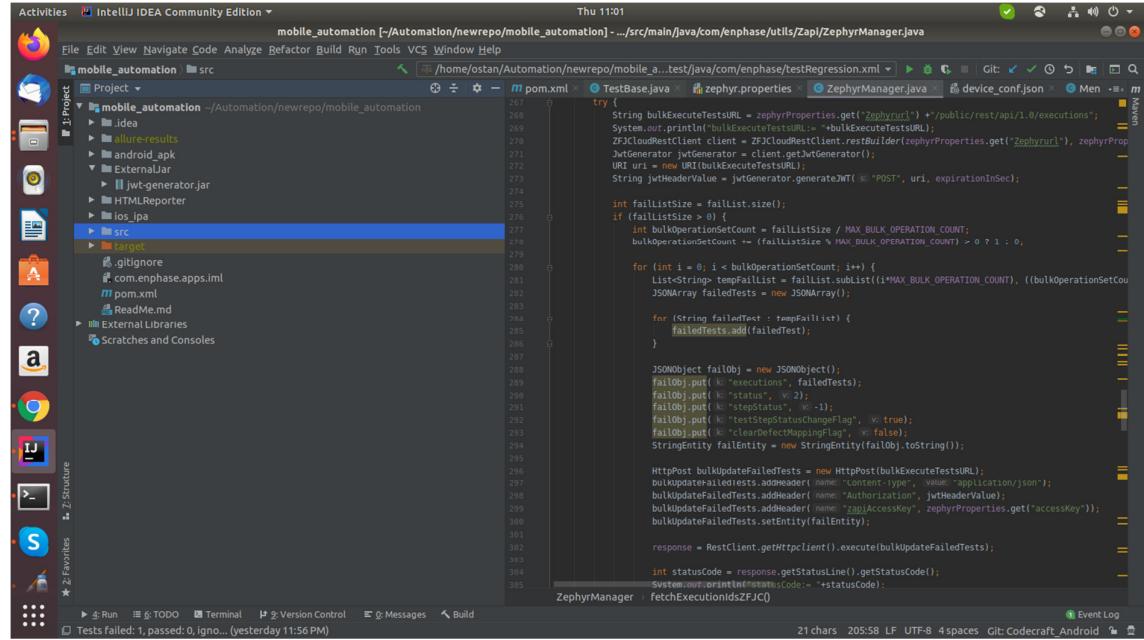
            response = RestClient.getHttpClient().execute(jobProgressRequest, RestClient.getContext(zephyrProperties));
            String result = EntityUtils.toString(response.getEntity());
            JSONParser parser = new JSONParser();
            HashMap<String, Object> json = HashMap<String, Object> parser.parse(result);

            String progress = String.valueOf(((JSONObject)json.get("progress")));
            if (progress != null && progress.equals("1.0")) {
                jobCompleted = true;
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
        System.out.println("jobCompleted: " + jobCompleted);
        return jobCompleted;
    }
}

```

Section 3.3: JWT Token

Above API calls make use of JWT token for authentication. So here we have made use of external jar provided by zephyr team for this purpose. You can download jar from [here](#)



```
try {
    String bulkExecuteTestsURL = zephyrProperties.get("zephyrurl") + "/public/rest/api/1.0/executions";
    System.out.println("bulkExecuteTestsURL=" + bulkExecuteTestsURL);
    ZF3CloudRestClient client = ZF3CloudRestClient.restBuilder(zephyrProperties.get("zephyrurl"), zephyrProperties.get("zephyrkey"));
    JwtGenerator jwtGenerator = client.getJwtGenerator();
    URI uri = new URI(bulkExecuteTestsURL);
    String jwtHeaderValue = jwtGenerator.generateJWT("POST", uri, expirationInSec);

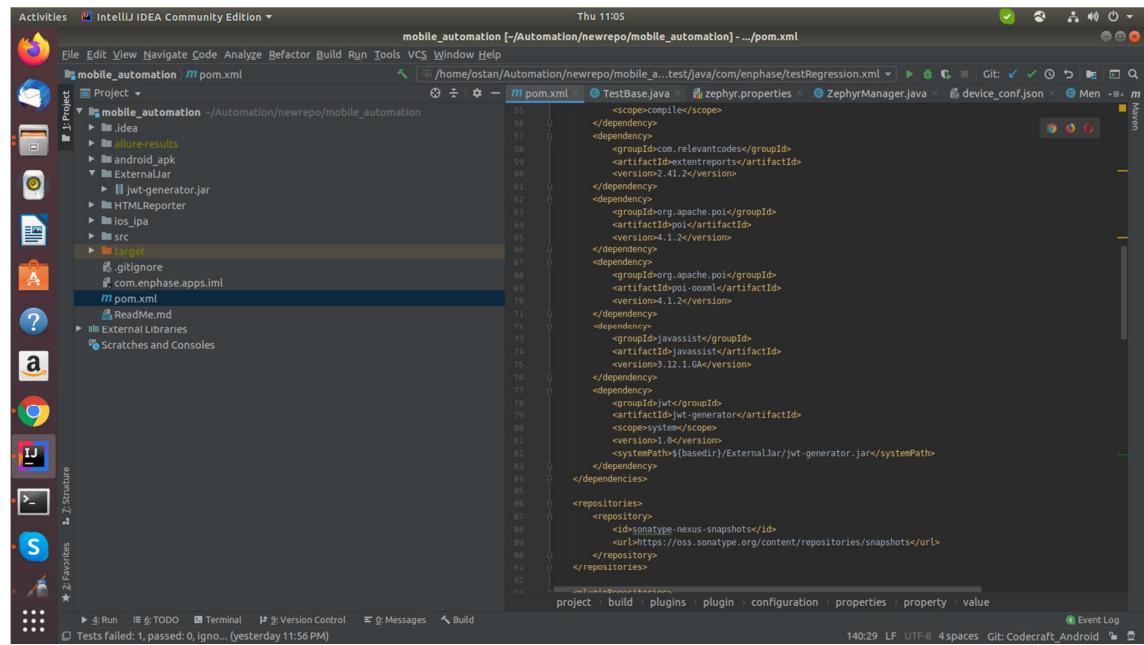
    int failListSize = failList.size();
    if (failListSize > 0) {
        int bulkOperationSetCount = failListSize / MAX_BULK_OPERATION_COUNT;
        bulkOperationSetCount += (failListSize % MAX_BULK_OPERATION_COUNT) > 0 ? 1 : 0;

        for (int i = 0; i < bulkOperationSetCount; i++) {
            List<String> tempFailList = failList.subList((i * MAX_BULK_OPERATION_COUNT), ((bulkOperationSetCount - 1) * MAX_BULK_OPERATION_COUNT));
            JSONObject failedTests = new JSONObject();
            for (String failedTest : tempFailList) {
                failedTests.add(failedTest);
            }

            JSONObject failObj = new JSONObject();
            failObj.put("executions", failedTests);
            failObj.put("status", "2");
            failObj.put("stepStatus", "-1");
            failObj.put("testStepExecutionChangeFlag", "true");
            failObj.put("testStepExecutionCopyFlag", "false");
            StringEntity failEntity = new StringEntity(failObj.toString());

            HttpPost bulkUpdateFailedTests = new HttpPost(bulkExecuteTestsURL);
            bulkUpdateFailedTests.addHeader("Content-type", "application/json");
            bulkUpdateFailedTests.addHeader("Authorization", jwtHeaderValue);
            bulkUpdateFailedTests.addHeader("zapiAccessKey", zephyrProperties.get("accessKey"));
            bulkUpdateFailedTests.setEntity(failEntity);

            response = RestClient.getHttpClient().execute(bulkUpdateFailedTests);
            int statusCode = response.getStatusLine().getStatusCode();
            System.out.println("statusCode=" + statusCode);
        }
    }
}
```



```
<dependency>
    <groupId>com.relevantcodes</groupId>
    <artifactId>extent-reports</artifactId>
    <version>2.41.2</version>
</dependency>
<dependency>
    <groupId>org.apache.poi</groupId>
    <artifactId>poi</artifactId>
    <version>4.1.2</version>
</dependency>
<dependency>
    <groupId>org.apache.poi</groupId>
    <artifactId>poi-ooxml</artifactId>
    <version>4.1.2</version>
</dependency>
<dependency>
    <groupId>javassist</groupId>
    <artifactId>javassist</artifactId>
    <version>3.12.1_GA</version>
</dependency>
<dependency>
    <groupId>jwt</groupId>
    <artifactId>jwt-generator</artifactId>
    <scope>system</scope>
    <version>1.0</version>
    <systemPath>${basedir}/ExternalJar/jwt-generator.jar</systemPath>
</dependency>
</dependencies>
<repositories>
    <repository>
        <id>sonatype-nexus-snapshots</id>
        <url>https://oss.sonatype.org/content/repositories/snapshots</url>
    </repository>
</repositories>
```

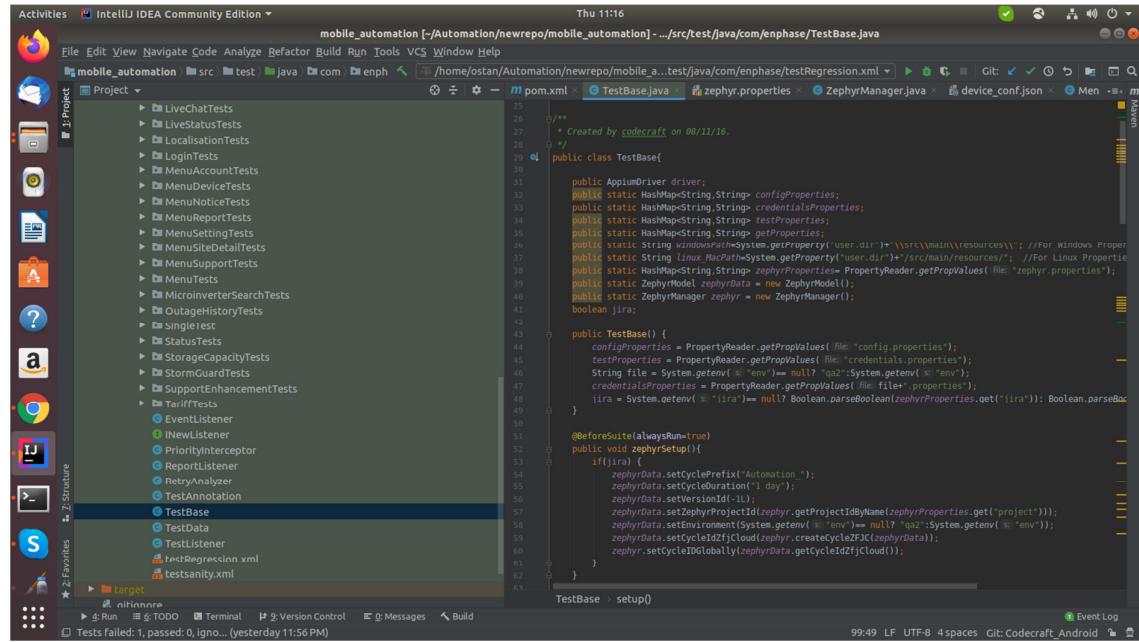
Chapter 4: Framework Changes

Below are the steps to be followed to integrate zephyr with automation framework.

Section 4.1: Zephyr Logic

a) Create Zephyr Cycle:

Zephyr cycle create should happen before test case execution starts. Hence we are making use of “**BeforeSuite**” annotation from TestNG for this purpose. And in response Cycle ID is stored in global variable for future use.



The screenshot shows the IntelliJ IDEA interface with the following details:

- File Bar:** Activities, IntelliJ IDEA Community Edition, mobile_automation [-/Automation/newrepo/mobile_automation] - .../src/test/java/com/enphase/testRegression.xml, Window Help.
- Pom.xml:** pom.xml
- Code Editor:** The file is TestBase.java. The code defines a class TestBase with a setup() method annotated with @BeforeSuite(alwaysRun=true). This method sets up a Zephyr cycle with a duration of 1 day and creates a cycle ID. It also sets the cycle prefix to "Automation_".

```
25
26 /**
27 * Created by codecraft on 08/11/16.
28 */
29 public class TestBase{
30
31     public AppiumDriver driver;
32     public static HashMap<String> configProperties;
33     public static HashMap<String> credentialsProperties;
34     public static HashMap<String> testProperties;
35     public static HashMap<String> getProperties();
36     public static String windowsPath=System.getProperty("user.dir")+"\\src\\main\\resources\\";
37     public static String Linux_MacPath=System.getProperty("user.dir")+"/src/main/resources/";
38     public static ZephyrModel zephyrData = new ZephyrModel();
39     public static ZephyrManager zephyr = new ZephyrManager();
40
41     boolean jira;
42
43     public TestBase() {
44         configProperties = PropertyReader.getPropValues( file: "config.properties");
45         testProperties = PropertyReader.getPropValues( file: "credentials.properties");
46         String file = System.getenv( &#039;env&#039;) == null? "qa2":System.getenv( &#039;env&#039;);
47         credentialsProperties = PropertyReader.getPropValues( file: "properties");
48         jira = System.getenv( &#039;jira&#039;) == null? Boolean.parseBoolean(ZephyrProperties.get("jira")) : Boolean.parseBoolean(ZephyrProperties.get("jira"));
49     }
50
51     @BeforeSuite(alwaysRun=true)
52     public void zephyrSetup(){
53         if(jira) {
54             zephyrData.setCyclePrefix("Automation_");
55             zephyrData.setCycleDuration("1 day");
56             zephyrData.setDeploymentId(zephyr.getProjectByName(ZephyrProperties.get("project")));
57             zephyrData.setDeploymentSystem.getenv( &#039;env&#039;) == null? "qa2":System.getenv( &#039;env&#039;);
58             zephyrData.setCycleIdFromCloud(zephyr.createCycleFromZC(zephyrData));
59             zephyr.setCycleIdGlobally(zephyrData.getCycleIdFromCloud());
60         }
61     }
62 }
```

- Toolbars and Panels:** Standard IntelliJ toolbars and panels like the Project, Editor, and Navigator are visible.
- Status Bar:** Shows the current time as Thu 11:16, file encoding as UTF-8, and Git status as Codecraft_Android.

b) TestName Parameter:

All testcases must have an testname parameter under @test annotation, which will have testcase ID for identification.

The screenshot shows the IntelliJ IDEA interface with the following details:

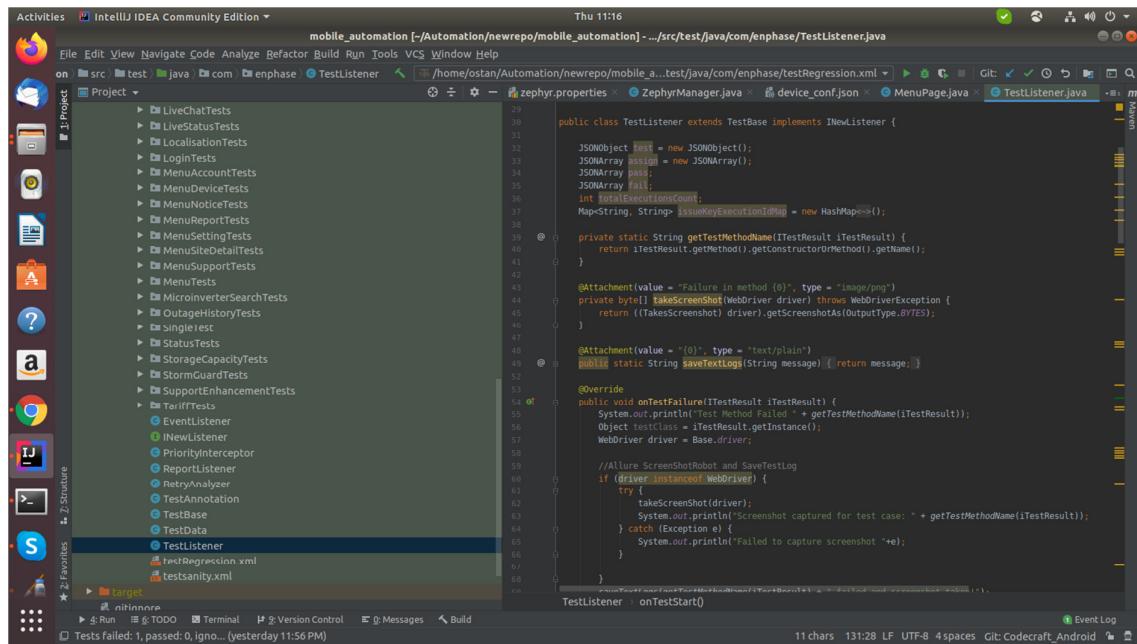
- Title Bar:** Activities → IntelliJ IDEA Community Edition - Thu 11:28
- Project Structure:** mobile_automation (~-/Automation/newrepo/mobile_automation) - .../src/test/java/com/enphase/ArrayTests/ArrayTests.java
- Code Editor:** The current file is ArrayTests.java, located at /home/ostan/Automation/newrepo/mobile_automation/src/test/java/com/enphase/testRegression.xml. The code implements several test cases for the ArrayPage:

 - `@Test(testName = "ENHO-1771") @Description("ENHO-1771 [Array Screen] - Verify different data filters") public void verifyDifferentDataFilters() { try { Pages.StatusPage().gotoStatusPage(); Assert.assertTrue(Pages.ArrayPage().verifyNavigatingToArrayScreen()); Assert.assertTrue(Pages.ArrayPage().verifyArrayFilters()); } catch (Exception e) { org.testng.Assert.fail("ENHO-1771 verifyDifferentDataFilters is failed\n" + e); } }`
 - `@Test(testName = "ENHO-1765") @Description("ENHO-1765 [Array Screen] - Verify Site Name") public void verifySiteName() { try { Pages.StatusPage().gotoStatusPage(); System.out.println("siteName=" + data.siteName()); Assert.assertTrue(Pages.ArrayPage().verifySiteName().contains(data.siteName())); } catch (Exception e) { org.testng.Assert.fail("ENHO-1765 verifySiteName is failed\n" + e); } }`
 - `@Test(testName = "ENHO-1763") @Description("ENHO-1763 [Array Screen] - Verify different layout selection") public void verifyTheDifferentLayoutSelection() { try { Pages.StatusPage().gotoStatusPage(); if (driver instanceof AndroidDriver) { ((AndroidDriver) driver).pressHome(); } else { driver.navigate().back(); } } catch (Exception e) { org.testng.Assert.fail("ENHO-1763 verifyTheDifferentLayoutSelection is failed\n" + e); } }`

- Bottom Status Bar:** Tests failed: 0, passed: 0, ignored: 0 (yesterday 11:56 PM)
- Event Log:** 53:54 LF UTF-8 4 spaces Git: Codecraft_Android

c) Making use of iTestListener Stages:

- **OnStart:** In this stage all declaration and initialization of variable that are going to be used further will happen in here.
 - **OnTestSuccess:** In this stage test under execution is added to the testcycle by using testname parameter and execution ID's for this test is also fetched and stored in array by name **Pass**.
 - **OnTestFailure:** In this stage test under execution is added to the testcycle by using testname parameter and execution ID's for this test is also fetched and stored in array by name **Fail**.
 - **OnTestSkipped:** In this stage test under execution is added to the testcycle by using testname parameter and execution ID's for this test is also fetched and stored in array by name **Skip**.
 - **OnFinish:** In this stage all test are executed based on array where we had stored test case ID.



```
Thu 11:16
mobile_automation [~/Automation/newrepo/mobile_automation] - .../src/test/java/com/enphase/TestListener.java
File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
Project Zephyr.properties ZephyrManager.java device_config.json MenuPage.java TestListener.java
Activities IntelliJ IDEA Community Edition
mobile_automation [~/Automation/newrepo/mobile_automation] - .../src/test/java/com/enphase/TestListener.java
File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
Project Zephyr.properties ZephyrManager.java device_config.json MenuPage.java TestListener.java
public class TestListener extends TestBase implements ITestListener {
    ...
    @Attachment(value = "Failure in method {0}", type = "image/png")
    private byte[] takeScreenshot(WebDriver driver) throws WebDriverException {
        return ((TakesScreenshot) driver).getScreenshotAs(OutputType.BYTES);
    }
    ...
    @Override
    public void onTestFailure(ITestResult iTestResult) {
        System.out.println("Test Method Failed " + getTestMethodName(iTestResult));
        Object testClass = iTestResult.getInstance();
        WebDriver driver = Base.driver;
        ...
        //Allure ScreenShotRobot and SaveTestLog
        if (driver instanceof WebDriver) {
            try {
                takeScreenshot(driver);
                System.out.println("Screenshot captured for test case: " + getTestMethodName(iTestResult));
            } catch (Exception e) {
                System.out.println("Failed to capture screenshot "+e);
            }
        }
    }
}
TestListener > onTestStart()
11 chars 131:28 LF UTF-8 4 spaces Git:Codecraft_Android Event Log
```