Integration of Jira and Xray with Carina Framework

Introduction:

This document serves as a guide for integrating Jira Xray with the Carina testing framework, enabling seamless defect management, test execution status updates, and evidence attachment within Jira Xray using Carina.

Prerequisites:

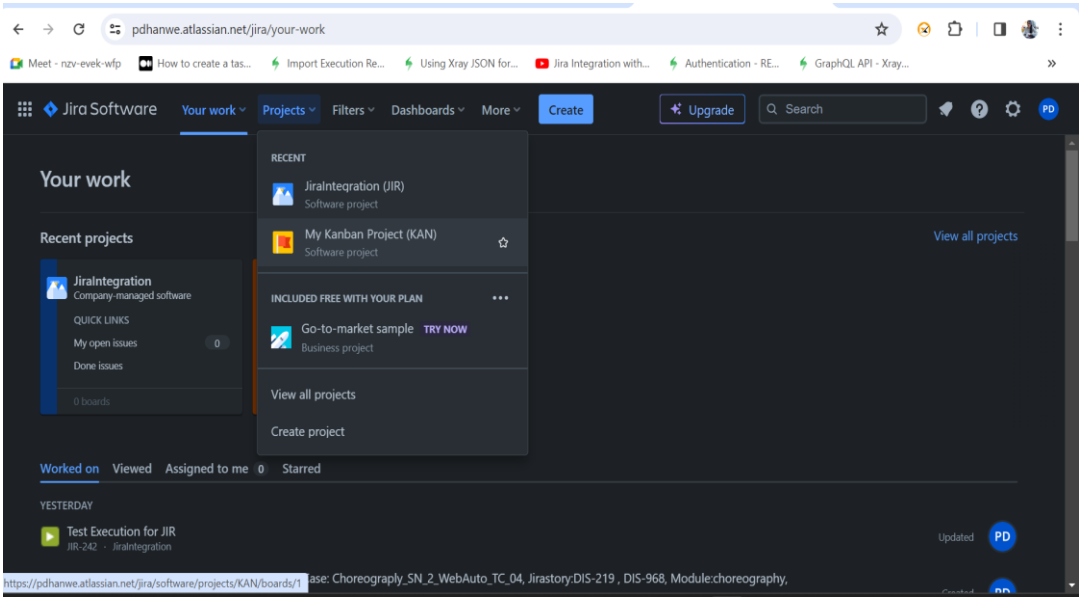
1. **Installation and configuration of Carina framework**:
   Ensure Carina framework is installed and configured properly.
2. **Access to Jira account with appropriate permissions**:
   Make sure you have access to a Jira account with appropriate permissions.
3. **Xray Plugin Configuration in Jira:**
   Ensure Xray plugin is configured in Jira.
4. **Required Maven dependencies**:
   Install necessary Maven dependencies.

Step1 - Configure Carina Project:

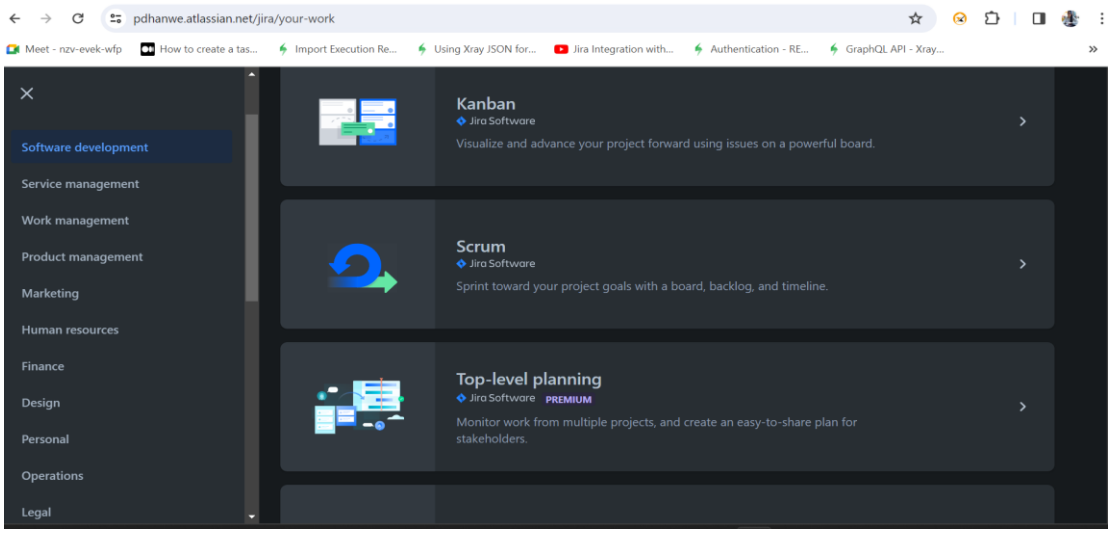
- Let's assume the Carina project is already configured and running

Step2 - Account Setup:

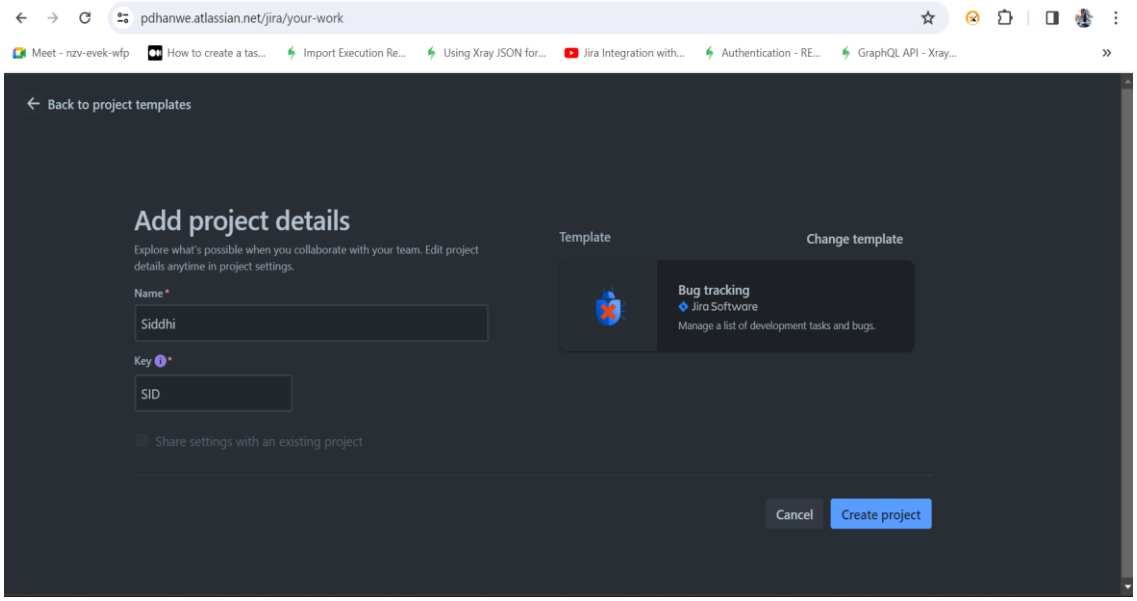
1. We should have Jira Account Access with UserID and Password
2. Login to Jira with your UserID and Password
3. Create a new project from Jira Software, Click on Projects->Create Project



4. Click on Software Development and Select theme from list kanban/bug tracking and follow instructions on screen. Click on Use Template

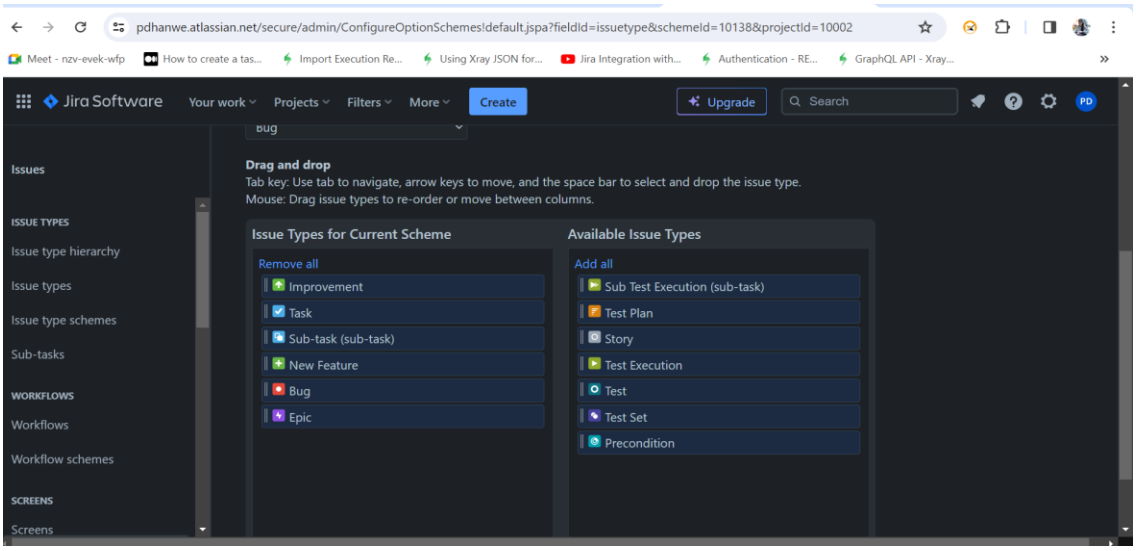


5. Add project Details, Type name of the project, project Key will get generated this key is important for integration, then click on Create project



6. From Project Settings Click on Summary->Software development Issue Type Scheme.

7. Select Actions->Edit Issue Types->Then drag and drop all the required issue types which we want and click on save.

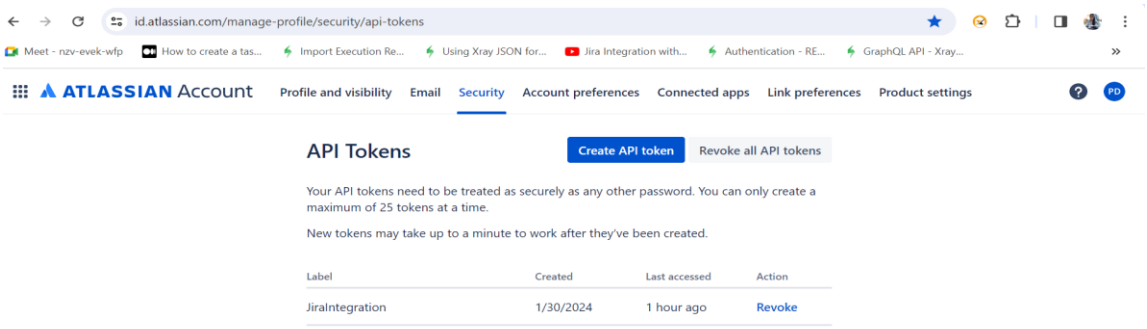


Step3 - Xray Plugin Configuration:

- Install Xray Plugin by following instructions on given link below: https://docs.getxray.app/display/XRAY/Installation

Step4 - Authentication:

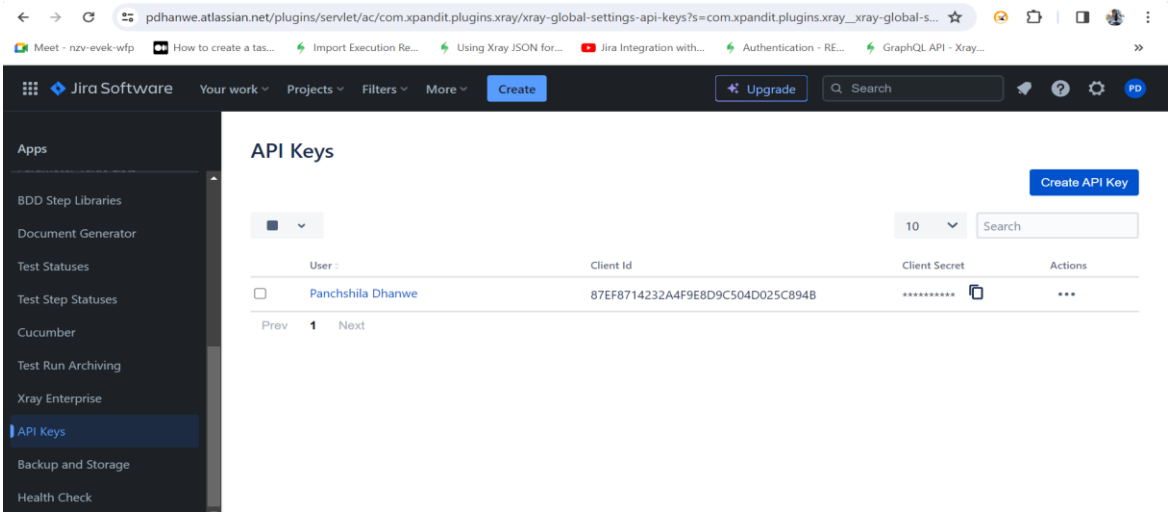
- **How To Generate API Token:**
  1. From Home Page Navigate to Profile->Account Settings->Security->Create and Manage API Tokens or Open this link - https://id.atlassian.com/manage-profile/security/api-tokens



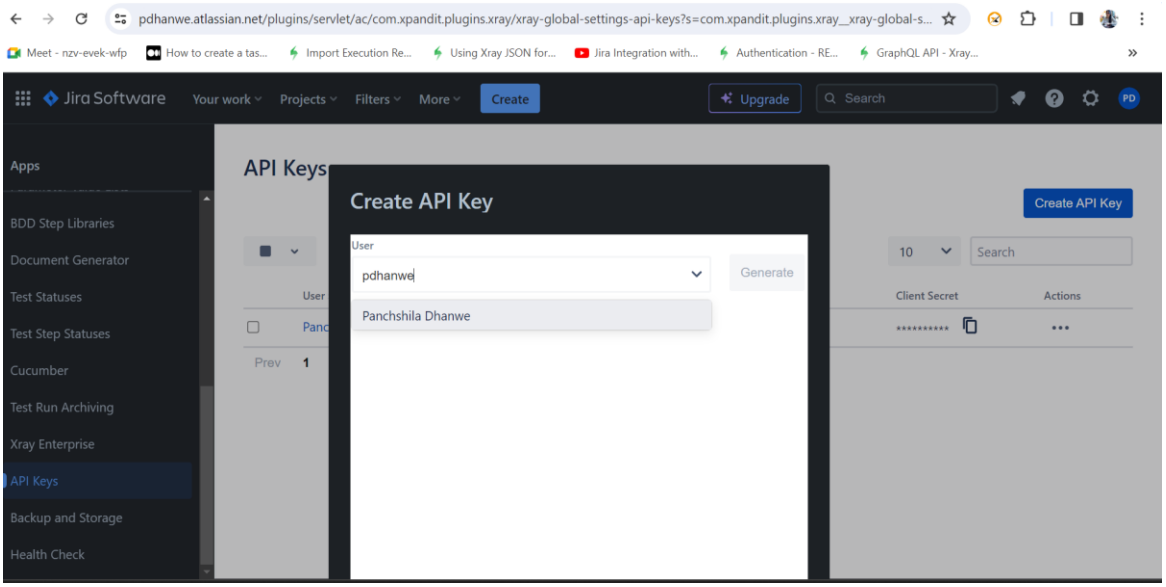
  2. Click on Create API Token
  3. Enter Label
  4. Click on Create
  5. Copy the API Token and Save it Somewhere

- **How To Generate Bearer Token:**

1. From Home Page Click on More->Apps->Manage Your Apps->Api Keys->Create Api Keys



2. Click on Create API Key
3. Type your name in user field

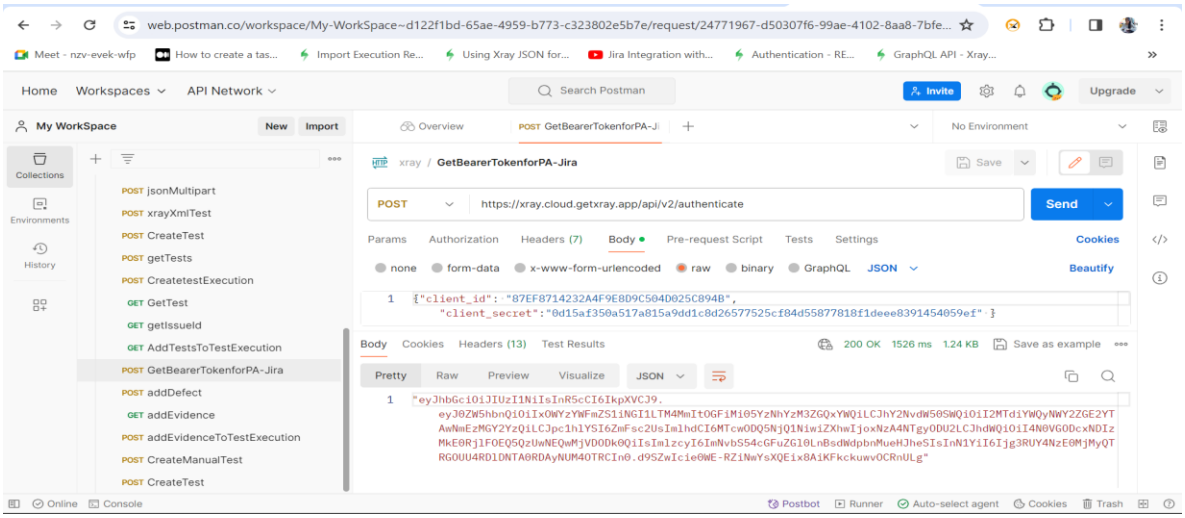


4. Then Click on Generate
5. Your Client_ID and Client_SECRET will get generated
6. We Will use this Client_id and Client_Scret for generating bearer token
7. We require Bearer Token for interacting with GraphQl API
8. We have created Bearer Token in two way one is manual using postman and another using Java through Automation
9. Bearer Token will get expired after 24 hrs, hence we have automated this part so that it will regenerate bearer token after 24 hrs and pass into our script

10. First, we will see how to generate this manually using postman
11. Open Postman, create Post Request and put the below endpoint in postman
    https://xray.cloud.getxray.app/api/v2/authenticate
12. Keep Body->raw>Json
13. Then Paste below code in Json Body

```
{"client_id":
"87EF8714232A4F9E8D9C504D0256674B","client_secret":"0d15af350a
517a815a9dd1c8d26577525cf84d55877818f1deee83665454059ef"}
```

14. Please replace this with your Client_ID and Client_Secret
15. Then Click on Send
16. You will see bearer token get generated in the response section with response code 200 like below.

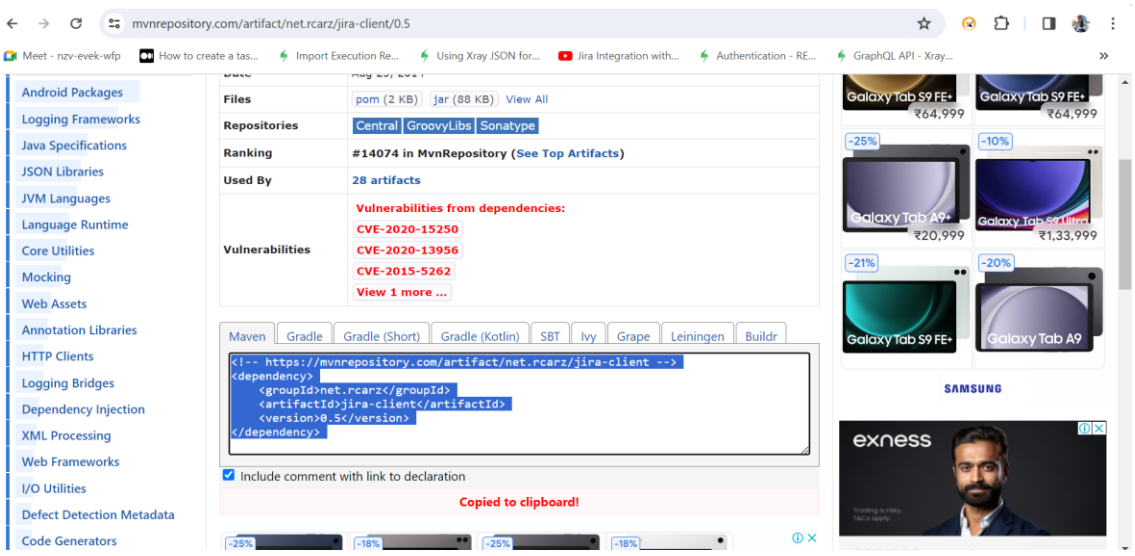


17. Copy and save the token somewhere

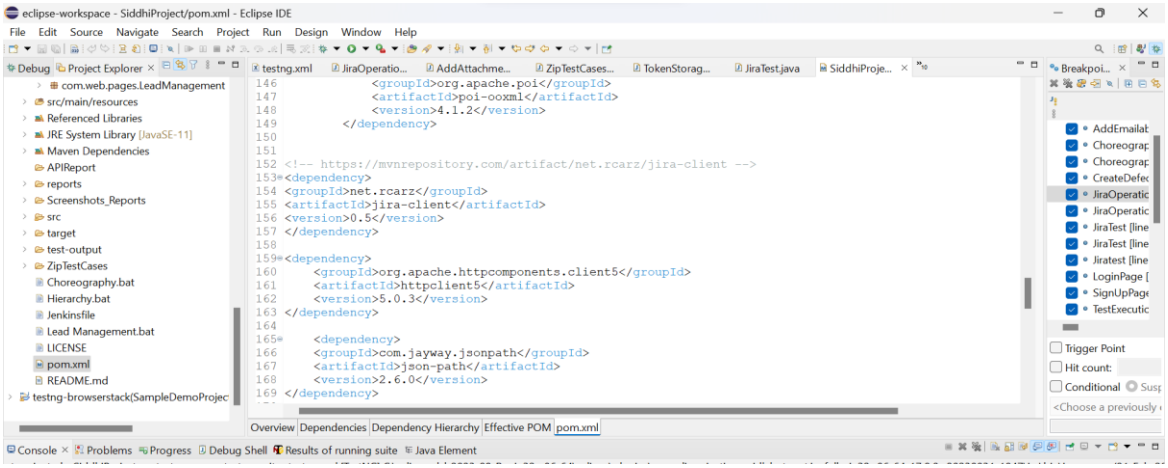
18. We Have Automated this part of generating Bearer token, hence you can skip this part if you want.

Step 5 - Required Dependencies:

1. For Jira Integration we will require below dependencies
2. Download the dependencies from maven repository and keep it in your pom file.



3. Copy the dependency and paste it in pom file.



4. Like this copy and paste all other dependencies and save it
5. After saving it will take few minutes to build the project and download the dependencies
6. if your pom file already contains some of the dependencies from below ones then please check, if the same dependencies are already available there, then no need to download it again
7. if we Copy and paste duplicate dependencies in pom file then it will create conflict in dependencies and can cause issues.

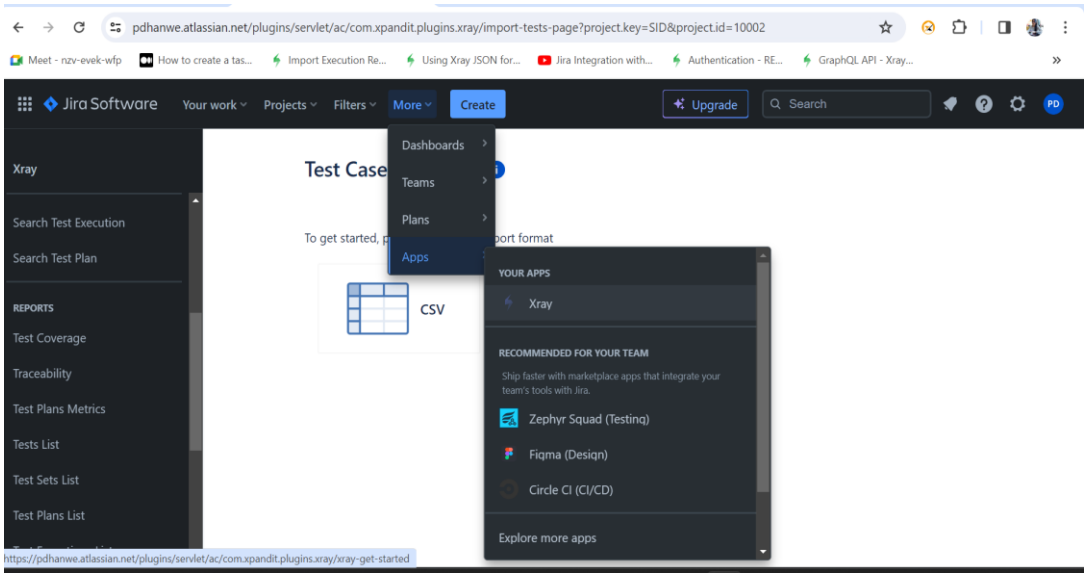
8. Below is the list of dependencies.

```
<!-- https://mvnrepository.com/artifact/com.googlecode.json-simple/jsonsimple-
->

            <dependency>
                <groupId>com.googlecode.json-simple</groupId>
                <artifactId>json-simple</artifactId>
                <version>1.1.1</version>
            </dependency>


<!-- https://mvnrepository.com/artifact/org.apache.poi/poi-ooxml -->
            <dependency>
                  <groupId>org.apache.poi</groupId>
                  <artifactId>poi-ooxml</artifactId>
                  <version>4.1.2</version>
            </dependency>


<!-- https://mvnrepository.com/artifact/net.rcarz/jira-client -->
            <dependency>
            <groupId>net.rcarz</groupId>
            <artifactId>jira-client</artifactId>
            <version>0.5</version>
            </dependency>

            <dependency>
                <groupId>org.apache.httpcomponents.client5</groupId>
                <artifactId>httpclient5</artifactId>
                <version>5.0.3</version>
            </dependency>

           <dependency>
               <groupId>com.jayway.jsonpath</groupId>
               <artifactId>json-path</artifactId>
               <version>2.6.0</version>
           </dependency>


            <dependency>
            <groupId>io.rest-assured</groupId>
            <artifactId>rest-assured</artifactId>
            <version>5.3.2</version>
            <scope>test</scope>
            </dependency>
```

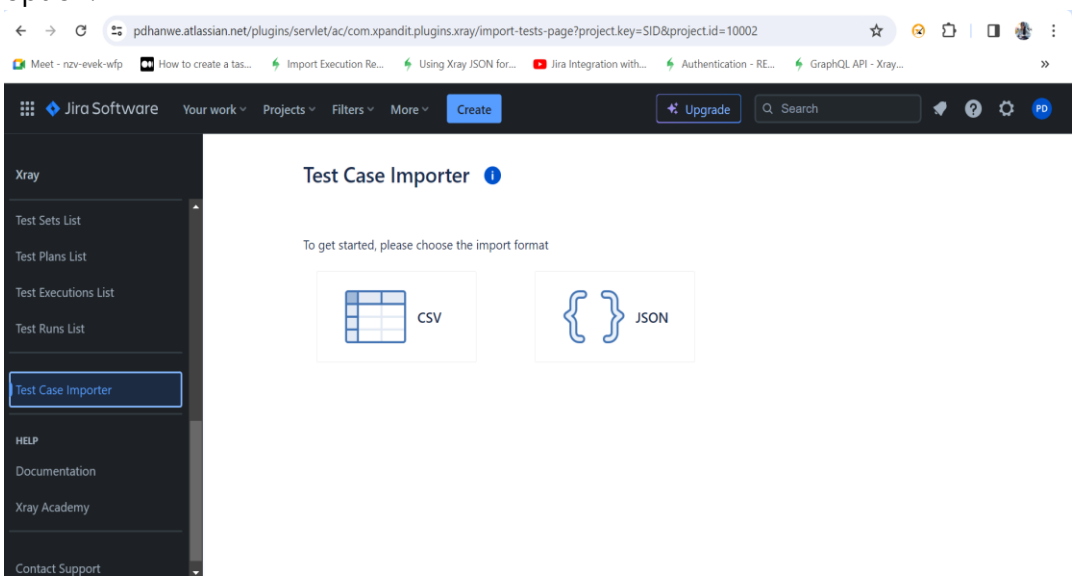
Alright! Now that we've covered all the necessary prerequisites Let's jump into the integration steps now.

Upload Test Cases in Xray:

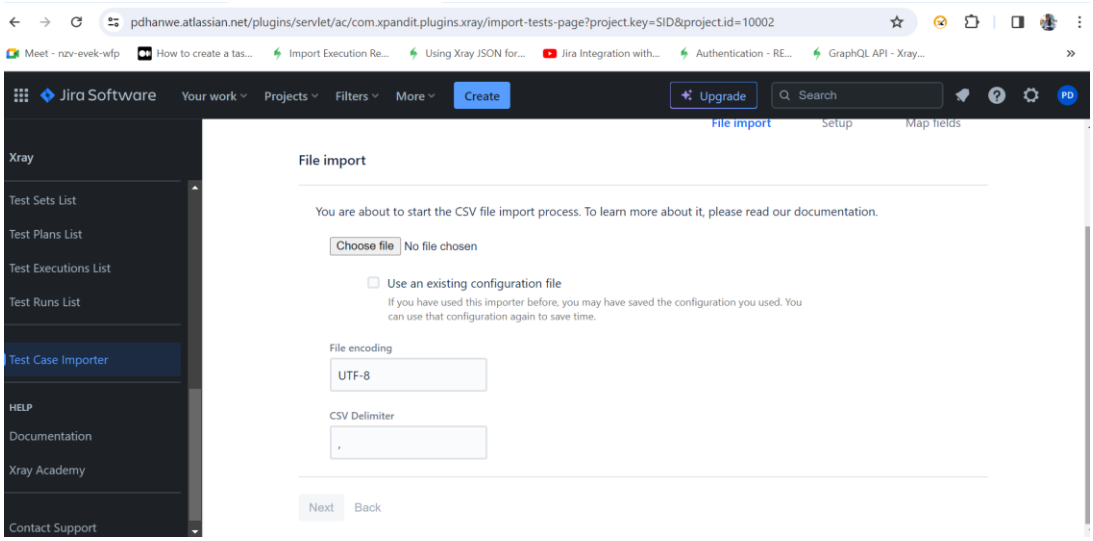
- Uploading testcases in Xray is Mandatory Step as we are going to associate our test results with these testcases.
- We should upload all required testcases in Xray which we are executing.
- Before uploading testcases in Xray we need to do some kind of formatting with testcases which is described in **How to format testcases to upload in Xray** Section below.
- Now we see Steps for **How to upload Testcases in Xray:**
    1. Click on More->Apps>Xray



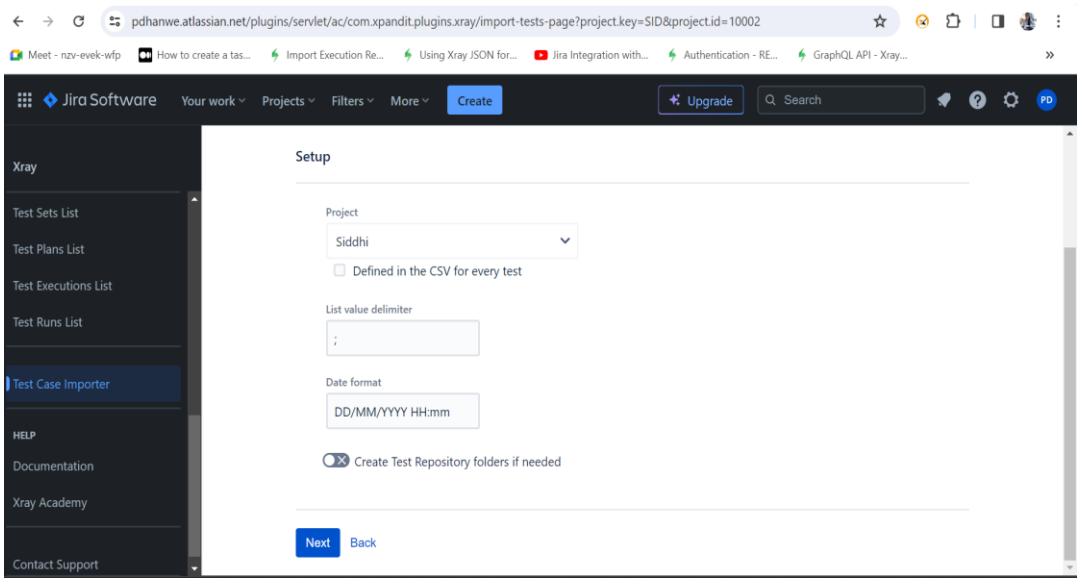
    2. From the left side panel scroll down and click on Test Case Importer option.



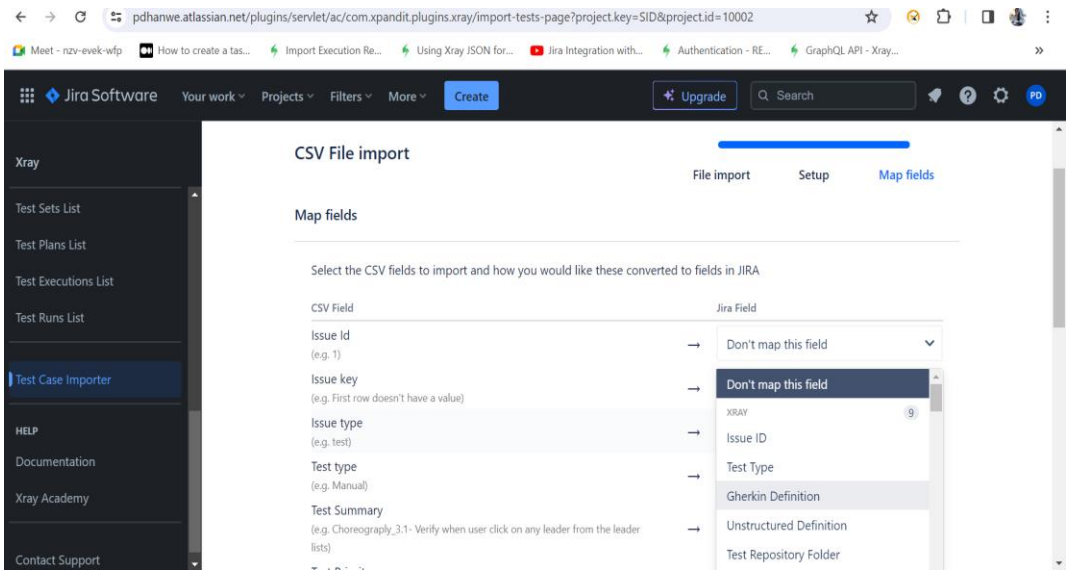
    3. Select .**csv** option from above screen



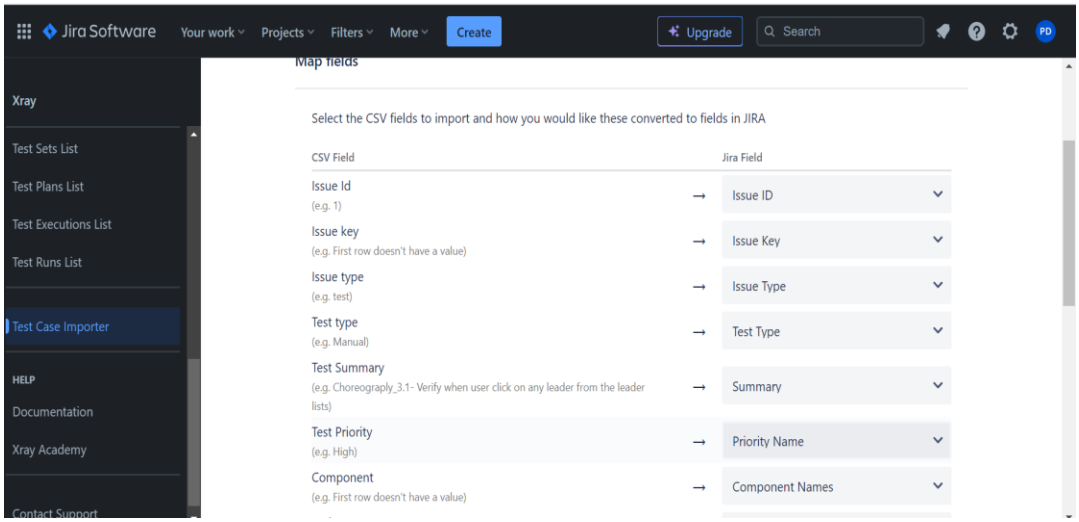
4. Click on Choose File and select the formatted .csv testcase file to upload
5. As we are uploading this first time, we will not select Use and Existing Configuration file option, we will use this option when next time we want to upload the testcases.
6. For now, click on next
7. From Next Screen Select the Project from dropdown and click on next

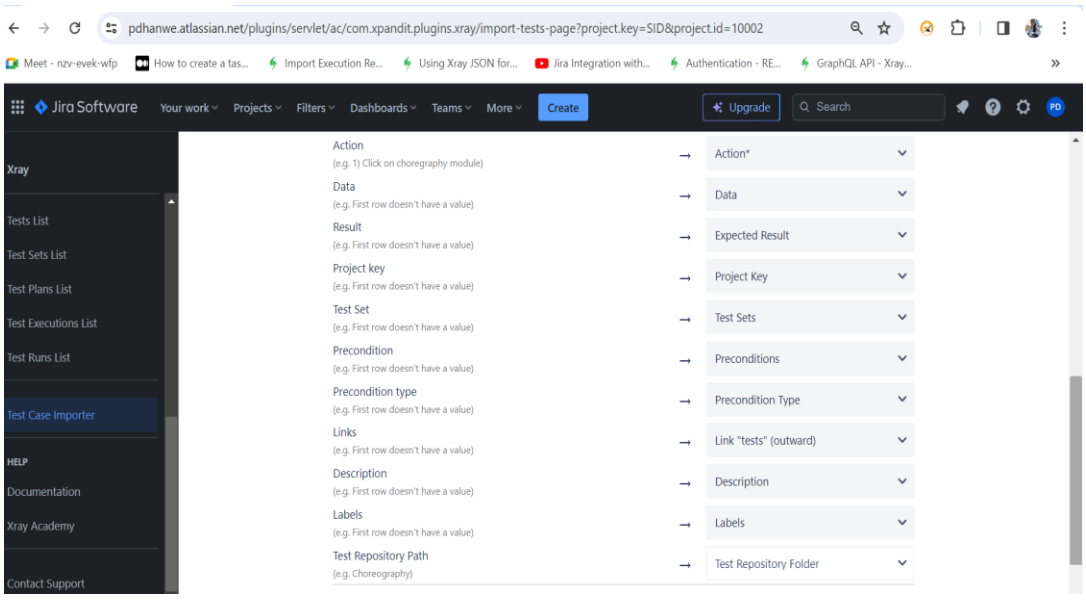


8. You will see below screen where we need to map the column names which are present in csv file with the Jira fields.
9. Please map the fields with the columns in csv as we have mapped in screenshot below.
10. In below screenshot header named CSV fields are column names present in csv file which we just uploaded we need to map that against the Jira fields which is present at right side from the dropdown.

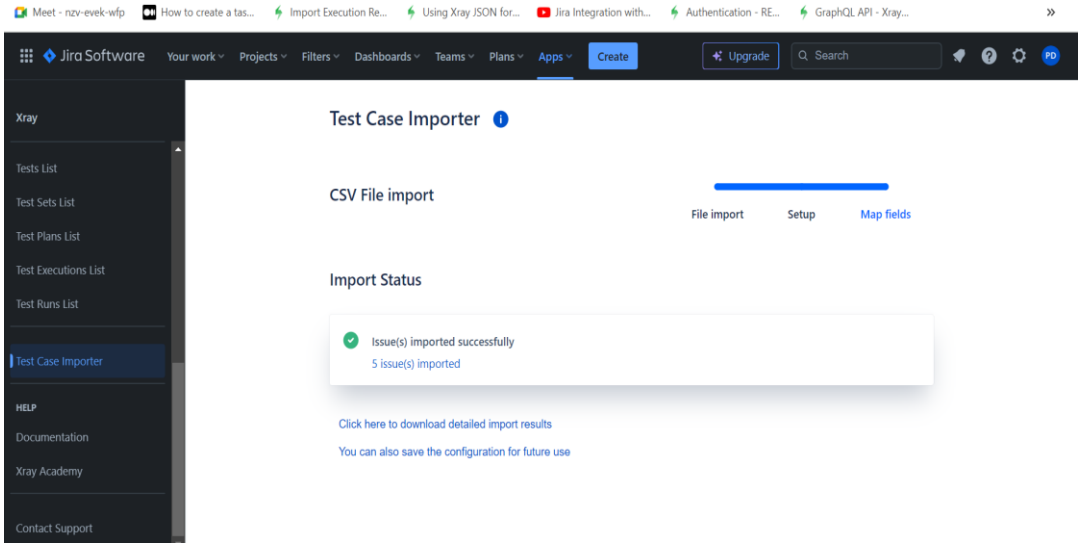


11. Map all the fields as shown below





12. If you have created test Repository already then map the filed to Test Repository folder otherwise select option of don't map this filed
13. After this click on next you will see below screen saying no. of issues imported



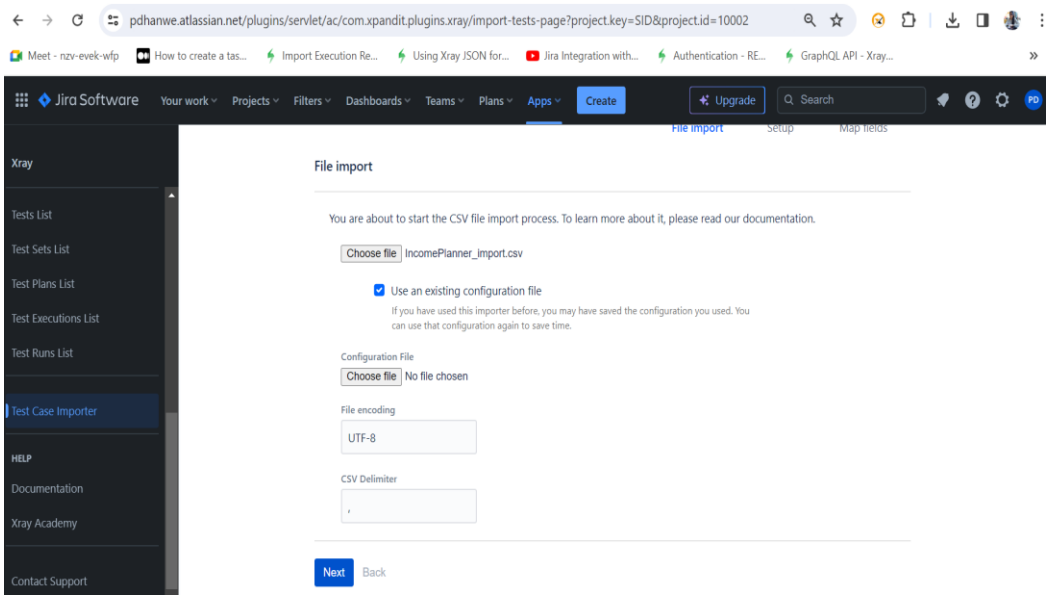
14. Now click on You can also save the configuration for future use

15. After this one config file will get downloaded which will be in json format

16. Save this file somewhere as this is important file for mapping the fileds

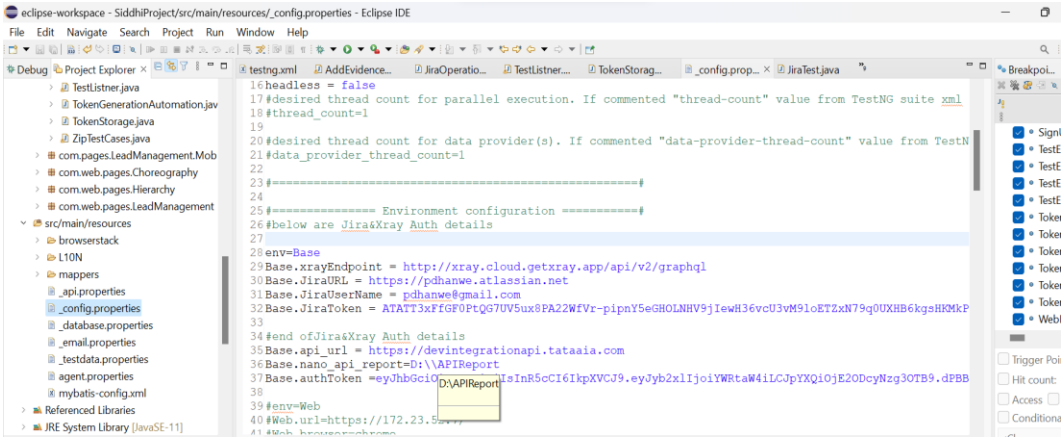
17. When we next time upload the testcases then check the option of 'use existing config file' so that we don't have need to map these fileds again manually, it will map all the fileds automatically with the help of config file.

18. Below is screenshot for adding config file , check the box and click on next



Integration Steps:

- **Step1: Configure Carina with Jira and Xray Credentials**: Update config file with authentication details
    1. In your project folder open **src/main/resources**
    2. Under which open **config.properties** file.
    3. In **config.properties** file add xray and Jira credentials like below



4. As present in above screenshot accordingly add your Auth details with respect to your account.
5. Here **Base.xrayEndPoint** is the endpoint for Xray GraphQL API
6. **Base.JiraURL** – is the URL for Jira Instance
7. **Base.JiraUserName** – is the username for accessing Jira
8. **Base.JiraToken –** is the API Token for authentication with Jira API
9. We will access these all Credentials in to our classes wherever required.

- **Step2: Let's Create Classes to integrate with Jira:**

Class1: JiraPolicy

1. Create package **com.jira.utils** in src/main/java
2. Create Class **JiraPolicy** in **com.jira.utils**
3. Copy and paste below code in JiraPolicy Class

```
package com.jira.utils;
import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;

@Retention(RetentionPolicy.RUNTIME)
public @interface JiraPolicy {

      boolean logTicketReady();
 }
}
```

Purpose of JiraPolicy Class:

- In JiraPolicy Class @Jirapolicy is custom annotation we have created. this class is indicating whether a Jira ticket should be logged for the annotated test method.
- It's not like every time we want to raise a ticket, so we need to maintain a flag, whenever that flag is true then only Create defect in jira.
- we can make it false as well @JiraPolicy (logTicketReady = false) - in case we don't want to associate the tests with Jira. so, it is providing a way to selectively decide which tests are associated with Jira and which ones are not. simply we can make it false in case if we don't want to create ticket in Jira.

Class2: JiraOperations

1. Create Class **JiraOperations** in **com.jira.utils**
2. Copy and paste below code in **JiraOperations Class**

```
package com.jira.utils;
import java.io.IOException;
import java.util.Base64;
import org.apache.http.HttpResponse;
import org.apache.http.client.HttpClient;
import org.apache.http.client.methods.HttpPost;
import org.apache.http.entity.StringEntity;
import org.apache.http.impl.client.HttpClientBuilder;
import org.apache.http.util.EntityUtils;
import org.json.simple.JSONObject;
import org.json.simple.parser.JSONParser;
import org.json.simple.parser.ParseException;
import com.zebrunner.carina.utils.R;
```

```
public class JiraOperations {

        String JiraURL=R.CONFIG.get("Base.JiraURL");
        String JiraUserName= R.CONFIG.get("Base.JiraUserName");
        String JiraToken= R.CONFIG.get("Base.JiraToken");

        //Create jira Issue as bug
        public String CreateJiraIssue(String projectName, String issueSummary, String
issueDescription,  String priority, String label, String assignee) throws IOException,
ParseException
        {
                String issueid = null; //to store issue/bug id
                HttpClient httpClient = HttpClientBuilder.create().build();

                String url = JiraURL+"/rest/api/3/issue";

                HttpPost postRequest = new HttpPost(url);
                postRequest.setHeader("Content-Type", "application/json");
                postRequest.setHeader("Authorization", "Basic " +
Base64.getEncoder().encodeToString((JiraUserName + ":" + JiraToken).getBytes()));
                StringEntity params = new
StringEntity(CreatePayLoadForCreateJiraIssue(projectName, issueSummary,
issueDescription,  priority, label, assignee));
                postRequest.setEntity(params);
                HttpResponse response = httpClient.execute(postRequest);

                //convert httpresponse to string
                String jsonString = EntityUtils.toString(response.getEntity());
                //convert String to json
                JSONParser parser = new JSONParser();
                JSONObject json = (JSONObject) parser.parse(jsonString);
                //extract issuekey from json
                issueid = (String) json.get("key");
                 if (response.getStatusLine().getStatusCode() == 201) {
             System.out.println("Defect created successfully. Issue key: " +
response.getEntity().getContent().toString());
         } else {
             System.out.println("Failed to create defect. Status code: " +
response.getStatusLine().getStatusCode());
         }

                return issueid;
        }

        private static String CreatePayLoadForCreateJiraIssue(String projectName, String
issueSummary, String issueDescription,
                        String priority, String label, String assignee) {

                return "{\r\n"
                                + "    \"fields\": {\r\n"
                                + "       \"project\":\r\n"
                                + "       {\r\n"
                                + "          \"key\": \""+projectName+"\"\r\n"
                                + "       },\r\n"
                                + "       \"summary\": \""+issueSummary+"\",\r\n"
                                + "       \"description\": {\r\n"
                                + "          \"type\": \"doc\",\r\n"
                                + "          \"version\": 1,\r\n"
                                + "          \"content\": [\r\n"
                                + "            {\r\n"
                                + "              \"type\": \"paragraph\",\r\n"
                                + "              \"content\": [\r\n"
                                + "                {\r\n"
                                + "                  \"type\": \"text\",\r\n"
                                + "                  \"text\":
\""+issueDescription+"\"\r\n"
                                + "                }\r\n"
                                + "              ]\r\n"
                                + "            }\r\n"
                                + "          ]\r\n"
                                + "        },\r\n"
                                + "    \"priority\": {\r\n"
                                + "      \"name\": \""+priority+"\"\r\n"
                                + "    },\r\n"
                                + "    \r\n"
                                + "    \"labels\": [\r\n"
                                + "      \"bugfix\",\r\n"
                                + "      \"blitz_test\"\r\n"
                                + "    ],\r\n"
                                + "    \"issuetype\": {\r\n"
                                + "        \"name\": \"Bug\"\r\n"
                                + "    },\r\n"
                                + "     \"assignee\": {\r\n"
                                + "      \"name\": \"Panchshila\"\r\n"
                                + "    }\r\n"
                                + "  \r\n"
                                + "    }\r\n"
                                + "}";
        }

}
```

Purpose of JiraOperations Class:

- ✓ This class facilitates interactions with Jira's REST API for creating Jira issues, specifically bugs.
- ✓ It initializes the Jira URL, username, and token using values from the Config.properties file which we defined earlier above.
- ✓ **CreateJiraIssue**: This method sends an HTTP POST request to the Jira API endpoint to create a new issue (bug). It takes parameters such as project name, issue summary, description, priority, label, and assignee. It returns the issue ID of the created bug.
- ✓ In above method we are using below Endpoint for creating new issues in Jira, JiraUrl We are taking from config file

```
JiraURL+"/rest/api/3/issue"
```

- ✓ **CreatePayLoadForCreateJiraIssue**: This private method constructs the JSON payload required for creating a Jira issue. It formats the parameters into the appropriate JSON structure expected by the Jira API.
- ✓ We can now call CreateJiraIsseue Method as per our requirement.
- ✓ Modify Payload as per your project requirement, update project key for your project and all other details.
- ✓ We are going to call this CreateJirsIssue method in TestListner Class

Class3: TokenGenerationAutomation

1. Create package **com.listner** in src/main/java
2. Create Class **TokenGenerationAutomation.java** in **com.listner**
3. Copy and paste below code in **TokenGenerationAutomation Class**

```
package com.listner;
import org.apache.http.HttpEntity;
import org.apache.http.HttpResponse;
import org.apache.http.client.HttpClient;
import org.apache.http.client.methods.HttpPost;
import org.apache.http.entity.StringEntity;
import org.apache.http.impl.client.HttpClients;
import org.apache.http.util.EntityUtils;
import com.auth0.jwt.JWT;
import com.auth0.jwt.exceptions.JWTDecodeException;
import com.auth0.jwt.interfaces.DecodedJWT;
import java.io.IOException;
import java.nio.charset.StandardCharsets;
import java.time.Instant;
import java.util.Date;
```

```
public class TokenGenerationAutomation {
    private static final String AUTH_ENDPOINT =
"https://xray.cloud.getxray.app/api/v2/authenticate";
    private static final String CLIENT_ID = "87EF8714232A4F9E8D9C504D025C894B";
    private static final String CLIENT_SECRET =
"0d15af350a517a815a9dd1c8d26577525cf84d55877818f1deee8391454059ef";
    private static String token;

    public static String generateOrRetrieveApiToken() {
        String storedToken = TokenStorage.readTokenFromFile();
        if (storedToken != null && !isTokenExpired(storedToken)) {
            return storedToken;
        } else {
            token = regenerateToken();
            TokenStorage.writeTokenToFile(token);
            return token;
        }
    }

    private static boolean isTokenExpired(String token) {
        try {
            // Decode the JWT token
            DecodedJWT jwt = JWT.decode(token);

            // Extract the expiration claim
            Date expiration = jwt.getExpiresAt();

            // Convert the expiration date to Instant
            Instant expirationInstant = expiration.toInstant();

            // Get the current time
            Instant currentTime = Instant.now();

            // Compare the expiration time with the current time
            return expirationInstant.isBefore(currentTime);
        } catch (JWTDecodeException e) {
            // Handle decoding exception
            e.printStackTrace();
            return true; // Treat decoding failure as token expired
        }
    }

    private static String regenerateToken() {
        // Create HttpClient
        HttpClient httpClient = HttpClients.createDefault();

        // Create HttpPost with URL
        HttpPost httpPost = new HttpPost(AUTH_ENDPOINT);

        // Set headers
        httpPost.setHeader("Content-Type", "application/json");

        // JSON request body
        String requestBody = "{\"client_id\": \"" + CLIENT_ID + "\", \"client_secret\":
\"" + CLIENT_SECRET + "\"}";

        try {
            // Set request body
            StringEntity requestEntity = new StringEntity(requestBody);
            httpPost.setEntity(requestEntity);

            // Execute the request
            HttpResponse response = httpClient.execute(httpPost);

            // Get the response body
            HttpEntity entity = response.getEntity();
            String responseString = EntityUtils.toString(entity,
StandardCharsets.UTF_8);

            // Extract the token from the response
            String token = extractTokenFromResponse(responseString);

            // You can save the token to storage for future use

            return token;

        } catch (IOException e) {
            // Handle the exception gracefully, perhaps by returning a default token or
rethrowing the exception
            e.printStackTrace();
            return null;
        }
    }
```

```
private static String extractTokenFromResponse(String responseString) {
        // Debugging: Print out the responseString to check its content
        System.out.println("Response String: " + responseString);

        // Remove the surrounding quotes from the response string
        responseString = responseString.replaceAll("^\"|\"$", "");

        try {
            // Decode the JWT token
            DecodedJWT decodedJWT = JWT.decode(responseString);

            // Extract the token from the decoded JWT
            token = decodedJWT.getToken();

            // Return the extracted token
            return token;
        } catch (JWTDecodeException e) {
            // If there's an error decoding the JWT, handle it gracefully
            System.err.println("Error decoding JWT: " + e.getMessage());
            return null;
        }
    }
}
```

Purpose of TokenGenerationAutomation Class:

- ✓ Bearer token we are retrieving from TokenGenerationAutomationClass.
- ✓ Bearer token which we generate manually is valid for only 24 hrs post that it will expire.
- ✓ **TokenGenerationAutomation Class** is responsible for checking if the token is active or not if the token is expired then it will generate new token by passing client id and client secret to the graphQL API.
- ✓ After generating token, we are storing the token in one text file using TokenStorgae Class.
- ✓ Make Sure to replace Client_ID and Client_Secret with you Client_ID and Client_Secret in above code
- ✓ By using above class, we will be able to retrieve bearer token dynamically.

Class4: TokenStorgae

1. Create Class **TokenStorgae** in **com.listner**
2. Copy and Paste Below Code in **TokenStorage** Class
3. Create **token.txt** file in src/test/resources/data_sources/token.txt

```java
package com.listner;

import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Paths;

public class TokenStorage {

       private static final String TOKEN_FILE_PATH =
System.getProperty("user.dir") + File.separator + "src" + File.separator +
            "test" + File.separator + "resources" + File.separator + "data_source"
+ File.separator + "token.txt";

    public static String readTokenFromFile() {
        try {
            // Check if the token file exists
            File tokenFile = new File(TOKEN_FILE_PATH);
            if (tokenFile.exists()) {
                String token = new
String(Files.readAllBytes(Paths.get(TOKEN_FILE_PATH)));
                return token;
            }
        } catch (IOException e) {
            // Handle file reading exception
            e.printStackTrace();
        }
        // If the file doesn't exist or is empty, return null
        return null;
    }

    public static void writeTokenToFile(String token) {
        try {
            FileWriter writer = new FileWriter(TOKEN_FILE_PATH);
            writer.write(token);
            writer.close();
        } catch (IOException e) {
            // Handle file writing exception
            e.printStackTrace();
        }
    }
}
```

Purpose of TokenStorgae Class:

- ✓ **TokenGenerationAutomation and TokenStorgae** Classes are integrated with each other
- ✓ After generating token, we are storing the token in one text file using TokenStorgae Class.
- ✓ With the help of these 2 classes we will be able to retrieve bearer token dynamically and no need to worry about token expiry.

Class5: CreateDefectReadData

4. Create Class **CreateDefectReadData** in **com.listner**
5. Copy and paste below code in **CreateDefectReadData** Class

```java
package com.listner;
import org.apache.poi.ss.usermodel.*;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.text.SimpleDateFormat;
import java.util.Calendar;
import java.util.Date;
import java.util.HashMap;
import java.util.Map;

public class CreateDefectReadData {

 public static Map<String, Map<String, String>> scenarioDataMap = new HashMap<>(); //
Declare scenarioDataMap as a public static field

    public void readReportFile() throws FileNotFoundException, IOException {
         // Get today's date
        Date today = Calendar.getInstance().getTime();
        SimpleDateFormat dateFormat = new SimpleDateFormat("dd-MMM-yyyy");
        String todayDate = dateFormat.format(today);

        String FILE_PATH = System.getProperty("user.dir") + File.separator +
"Screenshots_Reports" + File.separator + todayDate + File.separator + "Report.xls";
        String SHEET_NAME = "ExecutionReport";
        try (FileInputStream fis = new FileInputStream(FILE_PATH)) {
            Workbook workbook = WorkbookFactory.create(fis);
            Sheet sheet = workbook.getSheet(SHEET_NAME);

            if (sheet != null) {
                Row headerRow = sheet.getRow(0); // Assuming first row contains column
headers

                // Find the index of the "Scenario ID" column
                int scenarioIdIndex = -1;
                for (int j = 0; j < headerRow.getLastCellNum(); j++) {
                    String columnName = headerRow.getCell(j).getStringCellValue();
                    if (columnName.equals("AutomationTestcaseID")) {
                        scenarioIdIndex = j;
                        break;
                    }
                }

                // Read data from each row and store it in scenarioDataMap
                for (int i = 1; i <= sheet.getLastRowNum(); i++) {
                    Row row = sheet.getRow(i);

                    // Get the Scenario ID for the current row
                    String automationTestcaseId =
row.getCell(scenarioIdIndex).getStringCellValue();

                    // Create a new map to store data for this scenario
                    Map<String, String> scenarioData = new HashMap<>();

                    // Iterate through each column (except Scenario ID) and store data in
the map
                    for (int j = 0; j < headerRow.getLastCellNum(); j++) {
                        if (j != scenarioIdIndex) {
                            String columnName = headerRow.getCell(j).getStringCellValue();
                            Cell cell = row.getCell(j);
                            String cellValue = (cell == null) ? "" :
cell.getStringCellValue(); // Handle null cells
                            scenarioData.put(columnName, cellValue);
                        }
                    }

                    // Store the scenario data in scenarioDataMap
                   scenarioDataMap.put(automationTestcaseId, scenarioData);

                }
            } else {
                System.out.println("Sheet '" + SHEET_NAME + "' not found in the Excel
file.");
            }
        }
    }
```

Purpose of CreateDefectReadData Class:

- ✓ CreateDefectReadData Class Required for Creating Defect in Jira, we have Defined JiraOperations Class but we have not called it yet with the data required for creating defect.
- ✓ As we know we cannot define testcase failure reason prior to the execution and raise the defect directly in Jira.
- ✓ In order to make it dynamic we have tried to take summary and description of the testcase from the Report file which is getting generated and stored on particular path of the report post execution.
- ✓ By doing this we are able to capture defect summary and description at runtime and raise defect in Jira.

Class6: addEvidenceToTest

1. Create Class **addEvidenceToTest** in **com.listner**
2. Copy and Paste Below Code in **addEvidenceToTest** Class

```
package com.listner;
import java.io.File;
import java.io.IOException;
import java.nio.file.Files;
import java.util.Base64;
import org.apache.hc.client5.http.classic.methods.HttpPost;
import org.apache.hc.client5.http.impl.classic.CloseableHttpClient;
import org.apache.hc.client5.http.impl.classic.CloseableHttpResponse;
import org.apache.hc.client5.http.impl.classic.HttpClients;
import org.apache.hc.core5.http.ParseException;
import org.apache.hc.core5.http.io.entity.EntityUtils;
import org.apache.hc.core5.http.io.entity.StringEntity;
import org.json.JSONObject;
import com.zebrunner.carina.utils.R;
public class AddEvidenceToTest {

         private static final String xrayEndpoint = R.CONFIG.get("Base.xrayEndpoint");
    private static final String bearerToken = TokenStorage.readTokenFromFile();
    private static String zipFolderPath =
System.getProperty("user.dir")+File.separator+"ZipTestCases"+File.separator;  // Provide the actual folder
path
    private static String updateEvidence(String testRunId, String zipFileName, String base64EncodedZip) {
        return "mutation {\n" +
                "    addEvidenceToTestRun(\n" +
                "        id: \"" + testRunId + "\",\n" +
                "        evidence: [\n" +
                "            {\n" +
                "                filename: \"" + zipFileName + "\",\n" +
                "                mimeType: \"application/zip\",\n" +
                "                data: \"" + base64EncodedZip + "\"\n" +
                "            }\n" +
                "        ]\n" +
                "    ) {\n" +
                "        addedEvidence\n" +
                "        warnings\n" +
                "    }\n" +
                "}";

}
    public static void addEvidence(String testRunId, String testCaseId) throws ParseException {
        try {
            // Assuming the zip folder path is defined correctly and accessible
            File folder = new File(zipFolderPath);
            File[] zipFiles = folder.listFiles();

            if (zipFiles != null) {
                for (File zipFile : zipFiles) {
                    // Extract the test case ID from the zip file name
                    String zipFileName = zipFile.getName();
                    String extractedTestCaseId = zipFileName.replace(".zip", "");

                    // Check if the zip file name matches the current testCaseId
                    if (extractedTestCaseId.equals(testCaseId)) {
                        // Read the binary data of the zip file
                        byte[] zipData = Files.readAllBytes(zipFile.toPath());

                        // Encode the binary data as base64
                        String base64EncodedZip = Base64.getEncoder().encodeToString(zipData);

                        // Build the GraphQL query dynamically
                        String graphQLQuery = updateEvidence(testRunId, zipFileName, base64EncodedZip);
```

Purpose of addEvidenceToTest Class:

- ✓ **AddEvidenceToTest**, is responsible for adding evidence to a test run
- ✓ The class reads zip files from a specified folder and matches them with the provided test case ID and if match found it attach the evidence against that particular test in xray.

Class7: addEvidenceToTest

1. Create Class **ZipTestCases** in **com.listner**
2. Copy and Paste Below Code in **ZipTestCases** Class

```
package com.listner;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.text.SimpleDateFormat;
import java.util.Calendar;
import java.util.Date;
import java.util.zip.ZipEntry;
import java.util.zip.ZipOutputStream;

public class ZipTestCases {

    public void zipTestCases(String baseFolder, String destinationFolder) {
        File baseDir = new File(baseFolder);

        // Check if the base directory exists
        if (!baseDir.exists() || !baseDir.isDirectory()) {
            System.out.println("Base directory does not exist or is not a directory.");
            return;
        }

        // Get today's date
        Date today = Calendar.getInstance().getTime();
        SimpleDateFormat dateFormat = new SimpleDateFormat("dd-MMM-yyyy");
        String todayDate = dateFormat.format(today);

        // Iterate through date folders
        for (File dateFolder : baseDir.listFiles()) {
            if (dateFolder.isDirectory() && dateFolder.getName().equals(todayDate)) {
                // Iterate through module folders
                for (File moduleFolder : dateFolder.listFiles()) {
                    if (moduleFolder.isDirectory()) {
                        // Iterate through submodule folders
                        for (File subModuleFolder : moduleFolder.listFiles()) {
                            if (subModuleFolder.isDirectory()) {
                                // Iterate through testcaseid folders
                                for (File testCaseFolder : subModuleFolder.listFiles()) {
                                    if (testCaseFolder.isDirectory()) {
                                        // Zip the testcaseid folder
                                        zipFolder(testCaseFolder.getAbsolutePath(),
                                                destinationFolder + File.separator +
testCaseFolder.getName() + ".zip");
                                    }
                                }
                            }
                        }
                    }
                }
            }
        }
    }

    public void zipFolder(String sourceFolderPath, String zipFilePath) {
        try (FileOutputStream fos = new FileOutputStream(zipFilePath);
             ZipOutputStream zos = new ZipOutputStream(fos)) {

            File sourceFolder = new File(sourceFolderPath);
            zipFile(sourceFolder, sourceFolder.getName(), zos);

            System.out.println("Folder successfully zipped: " + zipFilePath);
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    public void zipFile(File fileToZip, String fileName, ZipOutputStream zipOut) throws IOException {
        if (fileToZip.isDirectory()) {
            for (File childFile : fileToZip.listFiles()) {
```

Purpose of addEvidenceToTest Class:

- ✓ **ZipTestCases** and **addEvidenceToTest** classes are integrated with each other
- ✓ These classes will create Zip files of executed Testcases Screenshot Folders and Keep it on Specified Path, it deletes existing zip files from destination path and update the new ones.

Class8: AddAttachmentToJira

1. Create Class **AddAttachmentToJira** in **com.listner**
2. Copy and Paste Below Code in **AddAttachmentToJira** Class

```
import com.zebrunner.carina.utils.R;
import org.apache.hc.core5.http.ParseException;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.nio.file.Paths;
import java.util.Base64;
import java.util.zip.ZipEntry;
import java.util.zip.ZipOutputStream;

public class AddAttachmentToJira {

	String JiraURL=R.CONFIG.get("Base.JiraURL");
	String JiraUserName= R.CONFIG.get("Base.JiraUserName");
	String JiraToken= R.CONFIG.get("Base.JiraToken");

    public static void addAttachmentToTestExecution(String testExecutionId,
String filePath) throws ParseException {
	    // Replace "YOUR_ISSUE_ID" with the actual Jira issue ID or key
	    String testExecutionIssueId = testExecutionId;

	    try (CloseableHttpClient httpClient = HttpClients.createDefault()) {
	        String url = JiraURL + "/rest/api/3/issue/" + testExecutionIssueId +
"/attachments";
	        HttpPost postRequest = new HttpPost(url);

	        // Set the authorization header
	        String authHeader = "Basic " +
Base64.getEncoder().encodeToString((JiraUserName + ":" + JiraToken).getBytes());
	        postRequest.setHeader("Authorization", authHeader);
	        postRequest.setHeader("X-Atlassian-Token", "nocheck");

	        // Build the multipart entity with the file attachment
	        MultipartEntityBuilder entityBuilder =
MultipartEntityBuilder.create();
	        entityBuilder.addPart("file", new FileBody(new File(filePath),
ContentType.APPLICATION_OCTET_STREAM,
Paths.get(filePath).getFileName().toString()));
	        postRequest.setEntity(entityBuilder.build());

	        // Execute the request
	        try (CloseableHttpResponse response =
httpClient.execute(postRequest)) {
	            int statusCode = response.getCode();
	            String responseBody = EntityUtils.toString(response.getEntity());

	            // Handle the response as needed
	            System.out.println("Response Code: " + statusCode);
	            System.out.println("Response Body: " + responseBody);
	        }
	    } catch (IOException e) {
	        e.printStackTrace();
	    }
    }
```

```
public static void createAndAttachZip(String sourceFolderPath, String
destinationFolderPath, String zipFileName, String testExecutionId) throws
IOException, ParseException {
        try (FileOutputStream fos = new FileOutputStream(destinationFolderPath +
File.separator + zipFileName);
             ZipOutputStream zos = new ZipOutputStream(fos)) {

            File sourceFolder = new File(sourceFolderPath);
            zipFile(sourceFolder, sourceFolder.getName(), zos);

            String zippedFilePath = destinationFolderPath + File.separator +
zipFileName;
            addAttachmentToTestExecution(testExecutionId, zippedFilePath);
        }
    }

    private static void zipFile(File fileToZip, String fileName, ZipOutputStream
zipOut) throws IOException {
        if (fileToZip.isHidden()) {
            return;
        }
        if (fileToZip.isDirectory()) {
            File[] children = fileToZip.listFiles();
            for (File childFile : children) {
                zipFile(childFile, fileName + File.separator +
childFile.getName(), zipOut);
            }
        } else {
            try (FileInputStream fis = new FileInputStream(fileToZip)) {
                ZipEntry zipEntry = new ZipEntry(fileName);
                zipOut.putNextEntry(zipEntry);
                byte[] bytes = new byte[1024];
                int length;
                while ((length = fis.read(bytes)) >= 0) {
                    zipOut.write(bytes, 0, length);
                }
            }
        }
    }
}
```

Purpose of AddAttachmentToJira Class:

- ✓ **AddAttachmentToJira** Class is responsible for uploading overall execution report in Jira which contains result of all the testcases executed in one run.

Class8: TestListner

1. Now we will integrate and call all above classes in one class in order to achieve our Jira and Xray Integration
2. Create Class **TestListner** in **com.listner**
3. Copy and Paste Below Code in **TestListner** Class

```
package com.listner;
import java.io.IOException;
import java.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.Collection;
import java.util.Date;
import java.util.HashMap;
import java.util.List;
import java.util.Locale;
import java.util.Map;
import org.apache.hc.client5.http.classic.methods.HttpPost;
import org.apache.hc.client5.http.impl.classic.CloseableHttpClient;
import org.apache.hc.client5.http.impl.classic.CloseableHttpResponse;
import org.apache.hc.client5.http.impl.classic.HttpClients;
```

```
public class TestListner implements ITestListener {

        private static final String xrayEndpoint = R.CONFIG.get("Base.xrayEndpoint");
    private static final String bearerToken =
TokenGenerationAutomation.generateOrRetrieveApiToken();
    ZipTestCases zc =new ZipTestCases();

    static String testCaseId ;
    static String testRunid;
    static String firstIssueId;
    String emailableReportFilepath ;
    JiraPolicy jiraPolicy;

    private static HashMap<String, String> testCaseIdIssueIdMap = new HashMap<>(); //
Declaration here
    private Map<String, String> testResults = new HashMap<>();


    @Override
    public void onTestStart(ITestResult result){
        CheckJiraPolicy(result);
    }


    @Override
    public void onTestFailure(ITestResult result) {

        if (testCaseId != null) {
            testResults.put(testCaseId, "FAILED");
        } else {
            System.out.println("Test case ID not found for test: " + result.getName());
        }
    }

    @Override
    public void onTestSuccess(ITestResult result) {

       if (testCaseId != null) {
            testResults.put(testCaseId, "PASSED");
        } else {
            System.out.println("Test case ID not found for test: " + result.getName());
        }
    }

    @Override
    public void onTestSkipped(ITestResult result) {
        System.out.println("Test skipped: " + result.getMethod().getMethodName());
    }

    @Override
    public void onFinish(ITestContext context) {
        System.out.println("All tests finished");
        try {
                        sendTestResultsToXray();

                } catch (org.apache.hc.core5.http.ParseException | ParseException |
IOException e) {
                        // TODO Auto-generated catch block
                        e.printStackTrace();
                }
    }


 // Send test result To xray
    public void sendTestResultsToXray() throws org.apache.hc.core5.http.ParseException,
ParseException, IOException {
        // Call ZipTestcases
        startZipTC();

        // Fetch the first issue ID from the map
        firstIssueId = testCaseIdIssueIdMap.values().stream().findFirst().orElse(null);

        // Create and get the IssueId of the test execution
        String testExecutionId = createTestExecutionAndGetIssueId();
        System.out.println("Stored IssueId: " + testExecutionId);

        // Step 3-6: Add all other test issue IDs to the test execution
        for (String testIssueId : testCaseIdIssueIdMap.values()) {
            if (!testIssueId.equals(firstIssueId)) {
                addTestToTestExecution(testExecutionId, testIssueId);
            }
        }

        // Step 7: Get test run IDs for all test issue IDs
```

Purpose of TestListner Class:

- ✓ This class implements the **ITestListener** interface to listen to various test events in TestNG. It performs actions such as checking JIRA policy, updating test results, sending test results to Xray, creating defects in JIRA, and linking defects to test runs. It interacts with other classes and libraries to handle test execution, zip test cases, and manage attachments in JIRA.
- ✓ In TestListner Class we need to pass **BearerToken** and **XrayEndPoint** to integrate with Xray
- ✓ XrayEndPoint we will take from **config.properties** file
- ✓ **Expected Behaviour from TestListner Class.**
    - ♦ **Start Zip Test Cases**: Calls the `startZipTC` method to zip test cases from the specified source folder to the destination folder
    - ♦ **Create Test Execution and Get Issue ID**:
        1. Creates a test execution in Xray using a GraphQL mutation, specifying the test issue IDs and other details.
        2. Retrieves the issue ID of the created test execution.
    - ♦ **Add Tests to Test Execution**: Adds all test issues except the first one to the created test execution using a GraphQL mutation.
    - ♦ **Get Test Run IDs**: Retrieves test run IDs for all test issue IDs using a GraphQL query.
    - ♦ **Update Execution Status and Add Evidence**:
        1. Updates the execution status of each test run based on the test result (passed or failed).
        2. Adds evidence to the test run if the status is failed.
    - ♦ **Create Defect in Jira**:
        1. Creates a defect in Jira for failed test cases, extracting data from the test report.
        2. Links the created defect to the respective test run.
    - ♦ **Add XLS Report**: Adds the XLS report as an attachment to the test execution in Jira.
- ✓ Each step in this sequence contributes to the integration between test execution results and the Xray test management system, facilitating comprehensive reporting and defect tracking.

Suite: testing.xml

1. Finally Create **testing.xml** file under **testng_suites** in **src/test/resources**
2. Copy and paste below code in testing.xml file

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE suite SYSTEM "testng-1.0.dtd">
<suite name="JiraTest" parallel="none" thread-count="1">
    <parameter name="suiteOwner" value="qpsdemo" />
    <parameter name="zafira_project" value="UNKNOWN" />
     <listeners>
        <listener class-name="com.listner.TestListner" />
    </listeners>
    <test name="JiraIntegrationTest" parallel="none" thread-
count="1">
        <classes>
            <class name="com.web.JiraTest" />
        </classes>
    </test>
</suite>
```

- Now we need to call below two methods in our test class and assign **JiraPolicy annotation** to the test method
- In TestClass Call JiraPolicy Annotation as I have called in below code highlighted in blue
- Then Call **readIssueIdFromExcel** Method from TestListner Class and **readReportFile** method from CreateDefectReadData class.
- Called these Two methods in below class highlighted in blue

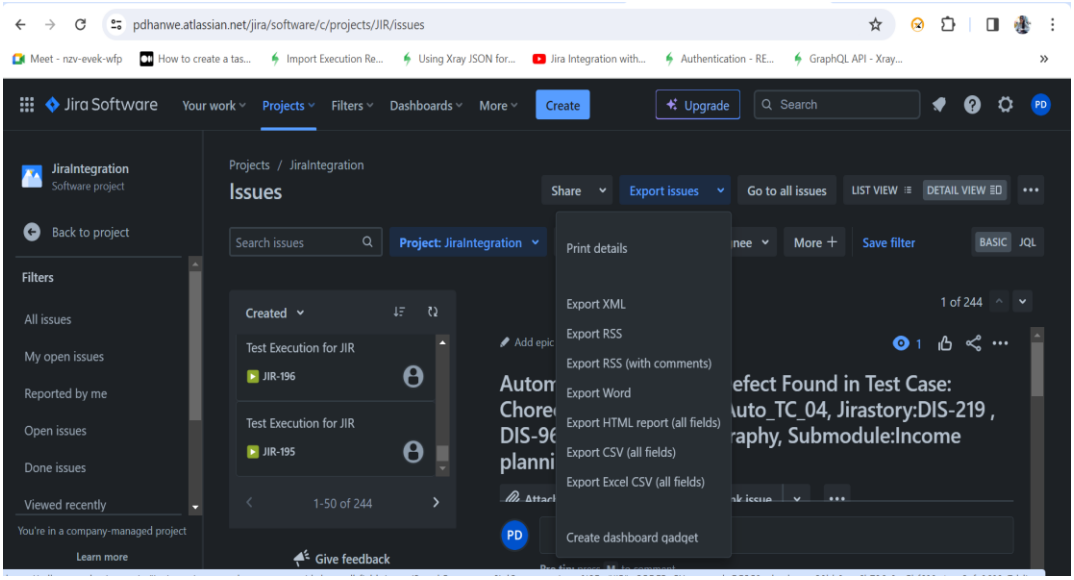
```
public class JiraTest implements IAbstractTest, IAbstractDataProvider {

      CreateDefectReadData cr = new CreateDefectReadData();
        CommonUtilities CU = new CommonUtilities();
        @JiraPolicy(logTicketReady = true)
        @Test(testName = "TC001", dataProvider = "DataPrivider")
        @XlsDataSourceParameters(path = "data_source/Siddhi_Web_TestCases.xlsx", sheet =
"Choreography", dsUid = "AutomationTestCaseID", executeColumn = "Executor", executeValue
= "Y")
public void Test(HashMap<String, String> args) throws Exception {

               try {
                       CU.FolderCreate();
                      CU.TCFolderCreate(args);
                      //Call page classes
                      TestListner tc = new TestListner();
                       tc.readTestIssueIdfromExcel(args);

                     HomePage home = new HomePage(getDriver());
                      home.openURL("", 100); //launch url


                  } catch (Exception ex) {
             ex.printStackTrace();
             args.put("status", "Fail");
             String actualResult = args.get("ActualResult");
             actualResult += "Test case Execution failed";
             args.put("ActualResult", actualResult);
             args.put("ExecutionResult", "Test Case failed due to error :  " +
ex.getMessage());
            if (args.get("Test Case Type").equalsIgnoreCase("Negative")) {
                args.put("status", "Pass");
            }
            CU.WriteToExcel(args);

            cr.readReportFile();

            throw new Exception(ex.getMessage());
```

- Alright, we have Defined all Required Classes Which are useful in integration with Xray.
- Now With Respect to Reading IssueId from testdata part we need to perform below steps.

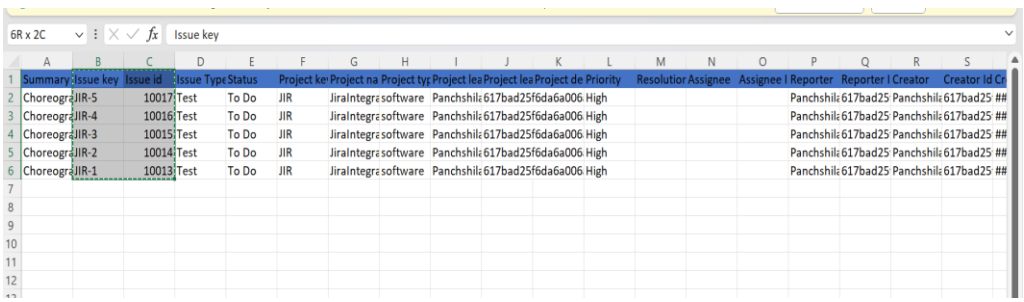
<u>Step3: Reading Test Issue Id's</u>:

Issue Id's are important point in our Integration part, based on the Isseid's we are able to perform all the necessary operations.

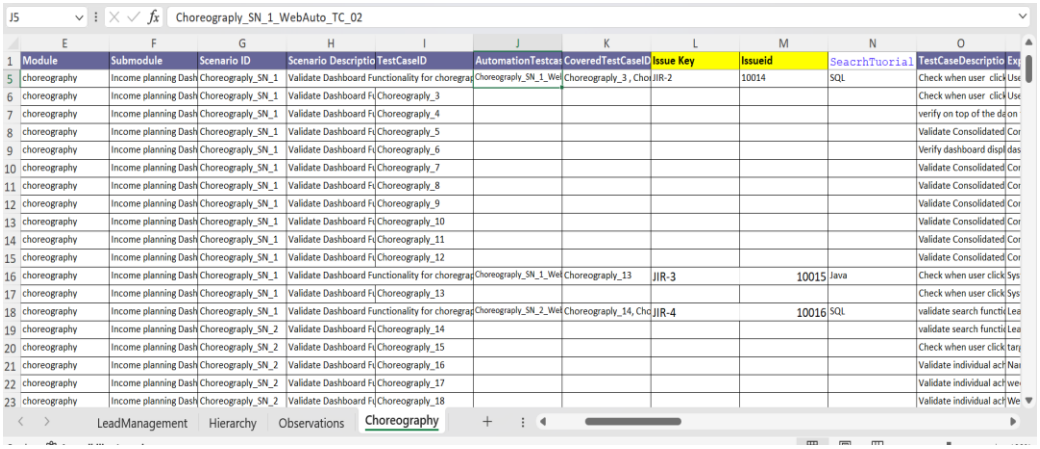
1. Issueid's for the Test we can get from Jira itself, assuming that we have Test's uploaded in Xray by Functional Team, if not we should upload the Test in xray to get the test issue id's.
2. In Jira go to our respective project and click on all issues, there you can see Export issues dropdown.
3. Click on the dropdown and Click on Export Excel CSV(all fields) option
4. After this file will get download which contains the issueid's for all the issuetypes which we created/uploaded.



5. Open the downloaded file and copy the issue ids for the test for which we want to update test status.



6. Copy and Paste the Issuekey and Issueid in your testdata file against the testcases we are running.



	E	F	G	H	I	J	K	L	M	N	O
1	Module	Submodule	Scenario ID	Scenario Descriptio	TestCaseID	AutomationTestcas	CoveredTestCaseID	Issue Key	Issueid	SeacrhTuorial	TestCaseDescriptio
5	choreography	Income planning Dash	Choreography_SN_1	Validate Dashboard Functionality for choregra		Choreography_SN_1_Wel	Choreography_3 , Cho	JIR-2	10014	SQL	Check when user click
6	choreography	Income planning Dash	Choreography_SN_1	Validate Dashboard Fu	Choreography_3						Check when user click
7	choreography	Income planning Dash	Choreography_SN_1	Validate Dashboard Fu	Choreography_4						verify on top of the da
8	choreography	Income planning Dash	Choreography_SN_1	Validate Dashboard Fu	Choreography_5						Validate Consolidated
9	choreography	Income planning Dash	Choreography_SN_1	Validate Dashboard Fu	Choreography_6						Verify dashboard displ
10	choreography	Income planning Dash	Choreography_SN_1	Validate Dashboard Fu	Choreography_7						Validate Consolidated
11	choreography	Income planning Dash	Choreography_SN_1	Validate Dashboard Fu	Choreography_8						Validate Consolidated
12	choreography	Income planning Dash	Choreography_SN_1	Validate Dashboard Fu	Choreography_9						Validate Consolidated
13	choreography	Income planning Dash	Choreography_SN_1	Validate Dashboard Fu	Choreography_10						Validate Consolidated
14	choreography	Income planning Dash	Choreography_SN_1	Validate Dashboard Fu	Choreography_11						Validate Consolidated
15	choreography	Income planning Dash	Choreography_SN_1	Validate Dashboard Fu	Choreography_12						Validate Consolidated
16	choreography	Income planning Dash	Choreography_SN_1	Validate Dashboard Functionality for choregra		Choreography_SN_1_Wel	Choreography_13	JIR-3	10015	Java	Check when user click
17	choreography	Income planning Dash	Choreography_SN_1	Validate Dashboard Fu	Choreography_13						Check when user click
18	choreography	Income planning Dash	Choreography_SN_1	Validate Dashboard Functionality for choregra		Choreography_SN_2_Wel	Choreography_14, Cho	JIR-4	10016	SQL	validate search functi
19	choreography	Income planning Dash	Choreography_SN_2	Validate Dashboard Fu	Choreography_14						validate search functi
20	choreography	Income planning Dash	Choreography_SN_2	Validate Dashboard Fu	Choreography_15						Check when user click
21	choreography	Income planning Dash	Choreography_SN_2	Validate Dashboard Fu	Choreography_16						Validate individual act
22	choreography	Income planning Dash	Choreography_SN_2	Validate Dashboard Fu	Choreography_17						Validate individual act
23	choreography	Income planning Dash	Choreography_SN_2	Validate Dashboard Fu	Choreography_18						Validate individual act

LeadManagement | Hierarchy | Observations | Choreography

7. This Will Make Sure that we are going to associate TestExecution Status against the respective testcases based on the issueid.

- Alright! We are now done with implementing all necessary classes, files and required test data
- Now Execute the `testing.xml` file to initiate the test execution
- After triggering the **testing.xml** file, all the automated processes will start working. handling test execution, updating pass/fail statuses in Xray, raising defects if tests fail, and managing evidence and attachments seamlessly.

✓ **Note:** To copy complete Class code Please Click into the Square box, Select all by pressing Ctrl+A and then Copy and paste in your Class

- **References:**
  - ✓ Jira-related operations, such as creating and updating issues, were performed using the Jira REST API. Further details can be found here - https://developer.atlassian.com/cloud/jira/platform/rest/v3/api-group-issues/#api-rest-api-3-events-get

  - ✓ Xray-related operations, including updating pass/fail statuses, raising defects, managing evidence, and attachments, were performed using the GraphQL API provided by Xray. More information can be found here- https://docs.getxray.app/display/XRAYCLOUD/GraphQL+API

    "By following these steps, you'll seamlessly integrate Jira and Xray, ensuring smooth test execution and accurate reporting for your projects."