GitHub Actions

In this document, we will show you how to put the unit, API and UI tests into a pipeline through GitHub Actions. GitHub allows us to run a series of commands on an event, such as push, merge or pull-request. A standalone command is called an action. You can create your own action or use and customize actions shared by the GitHub community. A task, which can execute an action is called a step. The step can also run shell commands. A series of steps is called a job. Workflows are .yml files consisting of one or more jobs and are stored in the .github/workflows directory. Unless said otherwise, the jobs in a workflow run in parallel. The environment, where they run, is called a runner. The runner can be either GitHub hosted or self-hosted. Diagram of a workflow can be seen in Fig. 1.

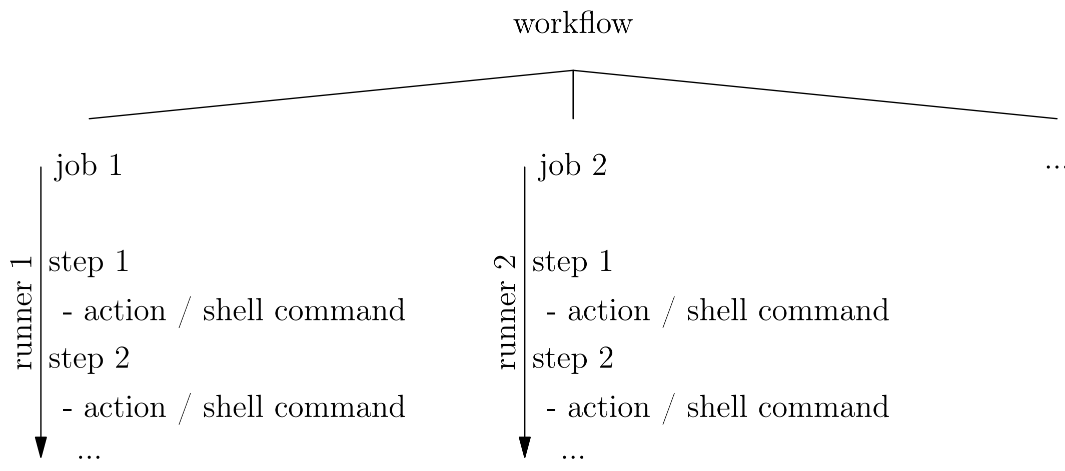
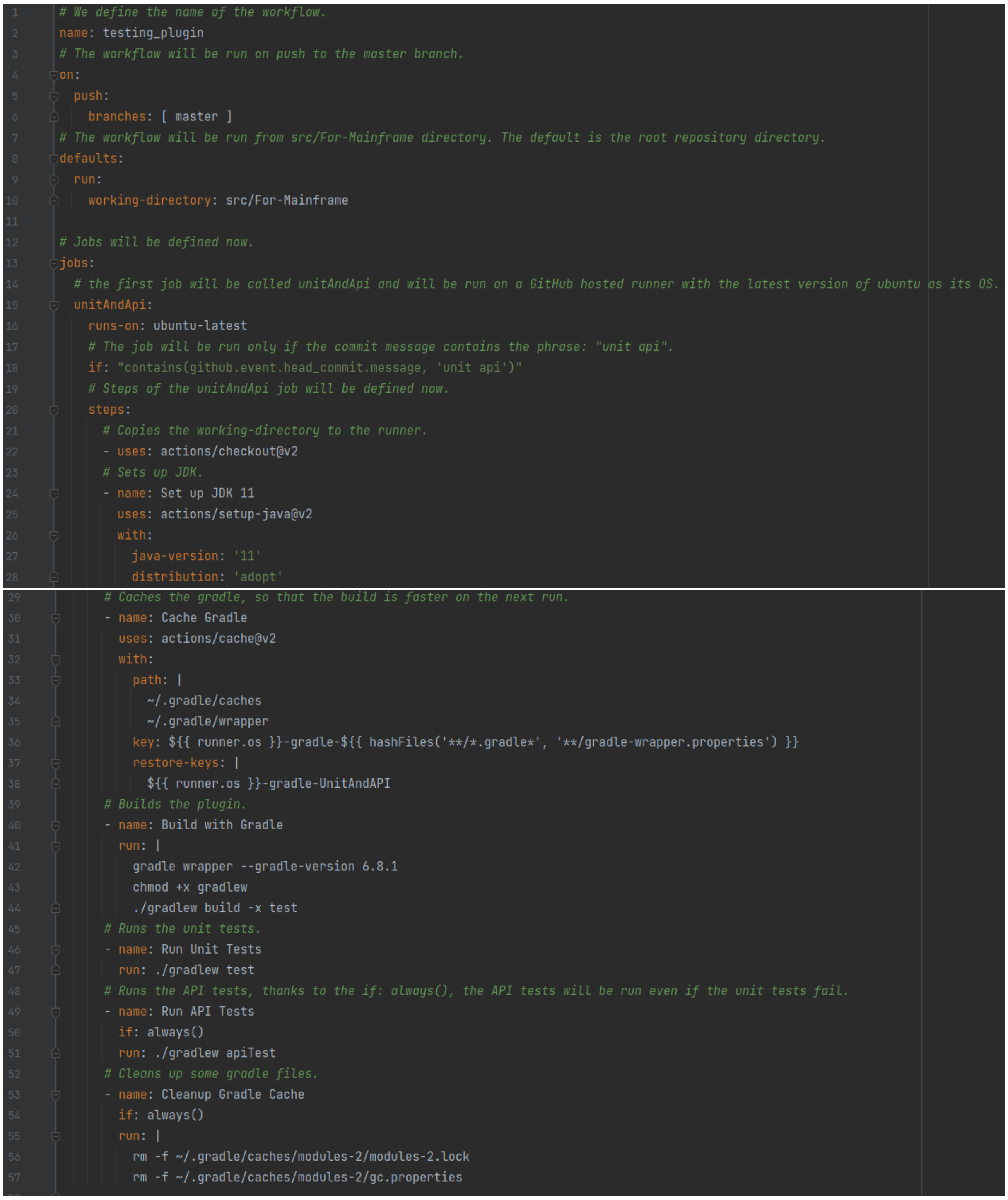


Fig. 1: Diagram of a workflow. The workflow contains jobs, which run in parallel. A job consists of a series of steps. Each step can run an action or a shell command.

In our case we used GitHub Actions to create a workflow, which runs the unit, API and UI tests. However, there were some issues with the UI tests and for now, they need a self-hosted runner to run. Also, I am not allowed to add self-hosted runners to the https://github.com/openmainframeproject-internship/Zowe-toolkit-plugin-for-Intellij repository. So, the workflow results, will be shown from this forked repository instead: https://github.com/MaliMi97/Zowe-toolkit-plugin-for-Intellij. Later in this document, we will tell you how to successfully run the pipeline.

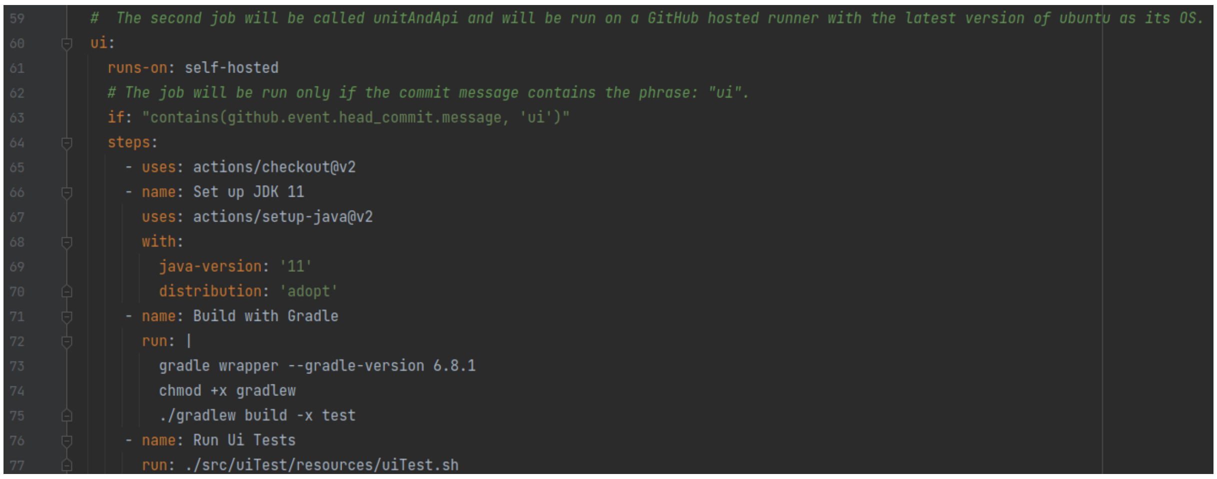
There is only one workflow in the repository. The workflow consists of two jobs. One, visible in Code 1, runs the unit and API tests. The other, visible in Code 2, runs the UI tests. Some actions, like action/setup-java@v2 or actions/cache@v2 have a clean up, which starts automatically at the end of the job. On lines 56 and 57 we remove some files, so that they are not being cached. The reason is that they might cause problems for future builds (https://docs.github.com/en/actions/guides/building-and-testing-java-with-gradle). In the ui job, Code 2, we do not cache the gradle at all, because we had troubles running the UI tests when we did.

By default in gradle, the unit tests are being run alongside the gradle build task. However, we run the unit tests separately. Also in GitHub Actions, unless said otherwise, once a step fails, the whole job fails and subsequent steps are not being run. We want the job to fail if the build of the plugin fails. But if the unit tests fail, we still want the API and UI test to run. Because of this, we build the plugin on lines 44 and 75 without unit tests. The unit tests are then run separately on line 47. Lastly, thanks to the line 50, the API tests will be run even if the unit tests fail.



```
1   # We define the name of the workflow.
2   name: testing_plugin
3   # The workflow will be run on push to the master branch.
4   on:
5     push:
6       branches: [ master ]
7   # The workflow will be run from src/For-Mainframe directory. The default is the root repository directory.
8   defaults:
9     run:
10      working-directory: src/For-Mainframe
11
12  # Jobs will be defined now.
13  jobs:
14    # the first job will be called unitAndApi and will be run on a GitHub hosted runner with the latest version of ubuntu as its OS.
15    unitAndApi:
16      runs-on: ubuntu-latest
17      # The job will be run only if the commit message contains the phrase: "unit api".
18      if: "contains(github.event.head_commit.message, 'unit api')"
19      # Steps of the unitAndApi job will be defined now.
20      steps:
21        # Copies the working-directory to the runner.
22        - uses: actions/checkout@v2
23        # Sets up JDK.
24        - name: Set up JDK 11
25          uses: actions/setup-java@v2
26          with:
27            java-version: '11'
28            distribution: 'adopt'
29        # Caches the gradle, so that the build is faster on the next run.
30        - name: Cache Gradle
31          uses: actions/cache@v2
32          with:
33            path: |
34              ~/.gradle/caches
35              ~/.gradle/wrapper
36            key: ${{ runner.os }}-gradle-${{ hashFiles('**/*.gradle*', '**/gradle-wrapper.properties') }}
37            restore-keys: |
38              ${{ runner.os }}-gradle-UnitAndAPI
39        # Builds the plugin.
40        - name: Build with Gradle
41          run: |
42            gradle wrapper --gradle-version 6.8.1
43            chmod +x gradlew
44            ./gradlew build -x test
45        # Runs the unit tests.
46        - name: Run Unit Tests
47          run: ./gradlew test
48        # Runs the API tests, thanks to the if: always(), the API tests will be run even if the unit tests fail.
49        - name: Run API Tests
50          if: always()
51          run: ./gradlew apiTest
52        # Cleans up some gradle files.
53        - name: Cleanup Gradle Cache
54          if: always()
55          run: |
56            rm -f ~/.gradle/caches/modules-2/modules-2.lock
57            rm -f ~/.gradle/caches/modules-2/gc.properties
```

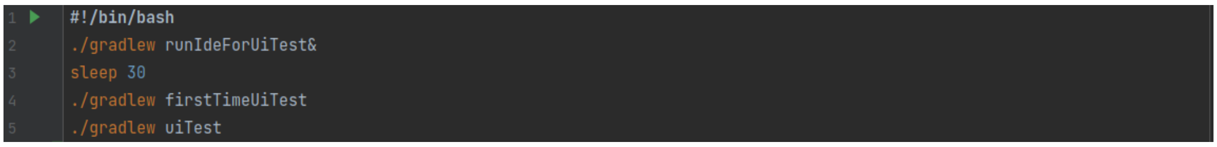
Code 1: The job running the unit and API tests.



```
59  #  The second job will be called unitAndApi and will be run on a GitHub hosted runner with the latest version of ubuntu as its OS.
60  ui:
61    runs-on: self-hosted
62    # The job will be run only if the commit message contains the phrase: "ui".
63    if: "contains(github.event.head_commit.message, 'ui')"
64    steps:
65      - uses: actions/checkout@v2
66      - name: Set up JDK 11
67        uses: actions/setup-java@v2
68        with:
69          java-version: '11'
70          distribution: 'adopt'
71      - name: Build with Gradle
72        run: |
73          gradle wrapper --gradle-version 6.8.1
74          chmod +x gradlew
75          ./gradlew build -x test
76      - name: Run Ui Tests
77        run: ./src/uiTest/resources/uiTest.sh
```

Code 2: The job running the UI tests.

Also, there are other issues with running UI tests in GitHub Actions. You can see on line 77 that we are running them through a shell script, visible in Code 3. Firstly, we run IDEA on the background. Then we wait for 30 sec. That is because it takes some time for the IDEA to start and we do not want the UI tests to run before it happens. On line 4 we run a special test, that accepts the user agreement. It is necessary to do so, since in each run, we start with a clean machine.



```
1  #!/bin/bash
2  ./gradlew runIdeForUiTest&
3  sleep 30
4  ./gradlew firstTimeUiTest
5  ./gradlew uiTest
```

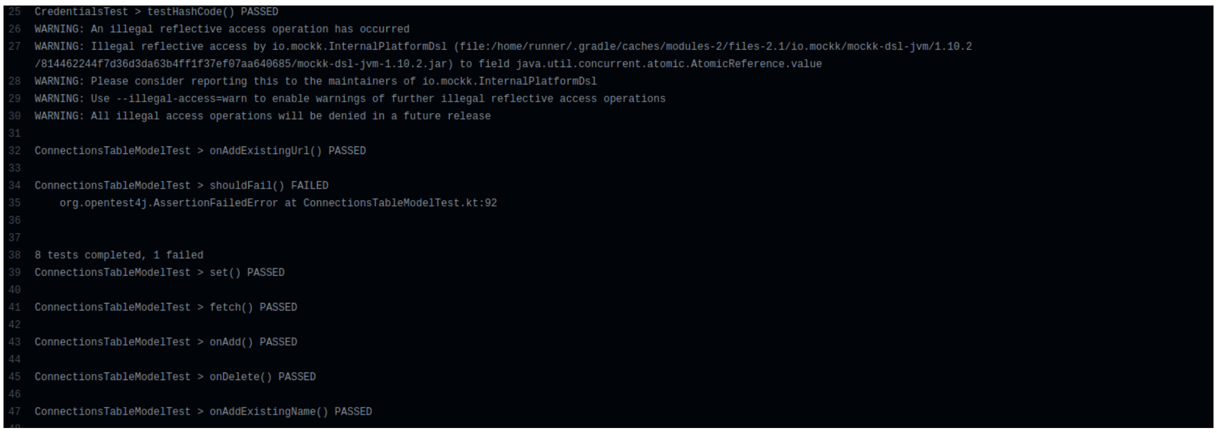
Code 3: The shell script for running UI tests.

Running the pipeline

You can run the pipeline by following these steps:

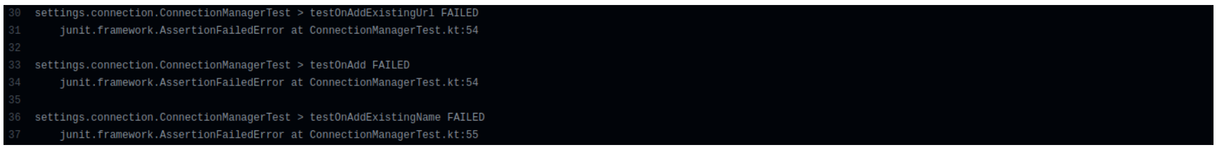
1. Fork the https://github.com/openmainframeproject-internship/Zowe-toolkit-plugin-for-Intellij repository.
2. In GitHub, go to Settings/Actions/Runners. Click on Add runner and follow the instructions. We used OS Linux and Architecture X64. There might be some issues if your runner will have something else.
3. Run the self-hosted runner if you have not done so already.
4. Make a small irrelevant change to the project and push a commit containing: "unit api ui" in its message.
5. Make sure that the IDEA, where the UI tests are being run is visible at all times.

The results can be seen here:
https://github.com/MaliMi97/Zowe-toolkit-plugin-for-Intellij/actions/runs/1197862927. By clicking on uniAndApi, we can see that firstly there are some warnings. Those are because of bugs



```
CredentialsTest > testHashCode() PASSED
WARNING: An illegal reflective access operation has occurred
WARNING: Illegal reflective access by io.mockk.InternalPlatformDsl (file:/home/runner/.gradle/caches/modules-2/files-2.1/io.mockk/mockk-dsl-jvm/1.10.2
/814462244f7d36d3da63b4ff1f37ef07aa640685/mockk-dsl-jvm-1.10.2.jar) to field java.util.concurrent.atomic.AtomicReference.value
WARNING: Please consider reporting this to the maintainers of io.mockk.InternalPlatformDsl
WARNING: Use --illegal-access=warn to enable warnings of further illegal reflective access operations
WARNING: All illegal access operations will be denied in a future release

ConnectionsTableModelTest > onAddExistingUrl() PASSED

ConnectionsTableModelTest > shouldFail() FAILED
    org.opentest4j.AssertionFailedError at ConnectionsTableModelTest.kt:92


8 tests completed, 1 failed
ConnectionsTableModelTest > set() PASSED

ConnectionsTableModelTest > fetch() PASSED

ConnectionsTableModelTest > onAdd() PASSED

ConnectionsTableModelTest > onDelete() PASSED

ConnectionsTableModelTest > onAddExistingName() PASSED
```

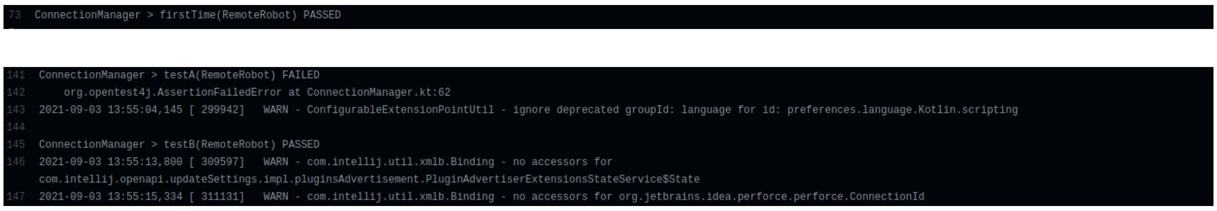
Fig. 2: Results of unit tests.



```
settings.connection.ConnectionManagerTest > testOnAddExistingUrl FAILED
    junit.framework.AssertionFailedError at ConnectionManagerTest.kt:54

settings.connection.ConnectionManagerTest > testOnAdd FAILED
    junit.framework.AssertionFailedError at ConnectionManagerTest.kt:54

settings.connection.ConnectionManagerTest > testOnAddExistingName FAILED
    junit.framework.AssertionFailedError at ConnectionManagerTest.kt:55
```

Fig. 3: Results of API tests.

in the plugin. Then, when we click on Run Unit Tests (Fig. 2), it is visible that all tests except the one that should fail (the shouldFail()) passed. The purpose for shouldFail was to introduce an error, so we know whether the if always() works or not. And it works, since the API tests have been run. Though all of the API tests failed (Fig. 3). The reason for that is being discussed in the document pertaining to the API tests. Only one UI tests failed (Fig. 4). The failure was introduced by us in order to check whether the tear down method works properly. The discussion about the tear down method can be seen in the document pertaining to the UI tests.



```
ConnectionManager > firstTime(RemoteRobot) PASSED
```

```
ConnectionManager > testA(RemoteRobot) FAILED
    org.opentest4j.AssertionFailedError at ConnectionManager.kt:62
2021-09-03 13:55:04,145 [ 299942]   WARN - ConfigurableExtensionPointUtil - ignore deprecated groupId: language for id: preferences.language.Kotlin.scripting

ConnectionManager > testB(RemoteRobot) PASSED
2021-09-03 13:55:13,800 [ 309597]   WARN - com.intellij.util.xmlb.Binding - no accessors for
com.intellij.openapi.updateSettings.impl.pluginsAdvertisement.PluginAdvertiserExtensionsStateService$State
2021-09-03 13:55:15,334 [ 311131]   WARN - com.intellij.util.xmlb.Binding - no accessors for org.jetbrains.idea.perforce.perforce.ConnectionId
```

Fig. 4: Results of UI tests.