API tests

In this section we will go over the approach of testing the For Mainframe plugin's API. For that we can use a by IntelliJ recommended way to test the plugin called modal level functional testing. The modal level functional tests run in a headless environment and use real implementation for most components. They test a feature as a whole, rather than individual functions and methods that comprise theirs implementation. Most of these tests in IntelliJ Platform codebase take a source file as an input data, execute a feature and then compare the output with expected result. The IntelliJ has build in classes methods, which help us with the implementation of the tests.

All of the model level functional tests need a project to be run in. This project is being set up along with the headless IDEA. There are two types of test classes provided by IntelliJ, Light and Heavy. Heavy test classes set up new project for every test. Light test classes reuse the project when possible. The set up is done in setUp method, the tear down in tearDown method. For the For Mainframe plugin, we propose to use the BasePlatformTestCase class, which is a Light test class and has no Java PSI functionality. We have set up another test case called PluginTestCase, which is a child of the BasePlatformTestCase and modifies the BasePlatformTestCase and allows us to API test the code. The classes, where the actual API tests of the plugin are written are children of the PluginTestCase.

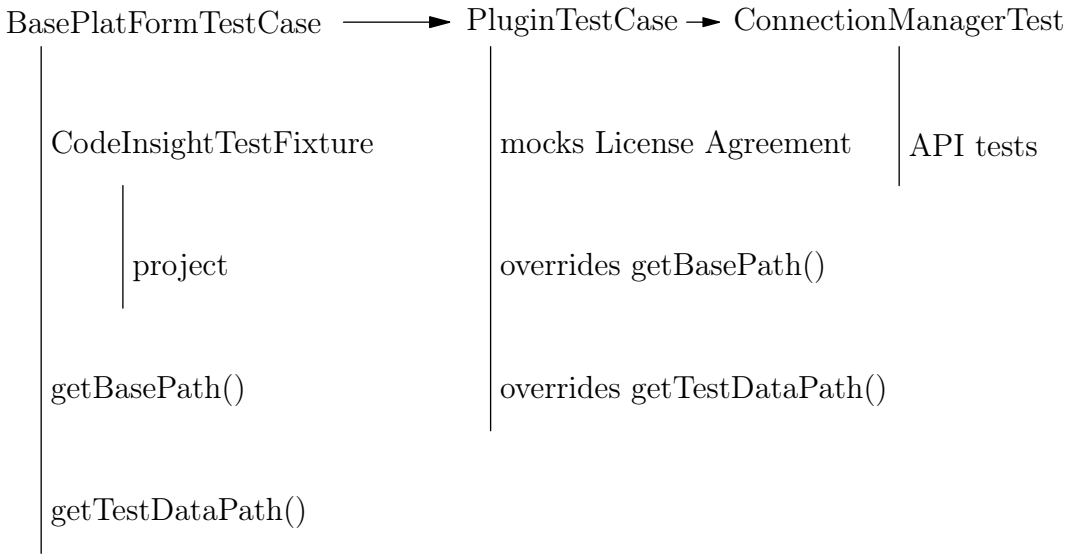

Fig. 1: The inheritance of API test cases along with some of their fields, methods or functionality. The ConnectionManagerTest class contains the API tests of the plugin.

In Figure 1 we can see that the BasePlatformTestCase class contains CodeInsightTestFixture and getBasePath, getTestDataPath methods. The CodeInsightTestFixture sets up and tears down the project, where the test is being run. The getBasePath and getTestDataPath methods provide relative and absolute path to a directory with testing data respectively. The testing data can be a file, which is taken as an input or used to compare the result.

The PluginTestCase, visible in Code 1, inherits the fields and methods of BasePlatformTestCase. The getBasePath and getTestDataPath methods are overriden. The For Mainframe plugin requires the user to agree to terms and conditions. In the plugin it is done via UI. In order to run any API test, there is a need to accept the terms and conditions in a headless IDEA. Because of that, the PluginTestCase modifies the setUp method and mocks the license agreement.

The ConnectionManagerTest is a child of the PluginTestCase and contains some API tests of the plugin. The tests address:

- Adding a connection to the plugin.
- Adding a connection with an existing name to the plugin.
- Adding a connection with an existing url to the plugin.

The test, which adds a connection to the plugin can be seen in Code 2. All tests check whether the variables sandboxCrudable and configCrudable have been modified accordingly. All of them fail. It seems that the procedures that delete the information from the two variables are not implemented properly. Here we can see that the testing approach has some values at it has already uncovered some possible bugs in the plugin. Developers of the plugin will be notified about the results of the tests.

```kotlin
/**
 * A custom test case for API tests.
 * This test case modifies the setUp method of BasePlatformTestCase and mocks the license agreement.
 * The class also overrides the getBasePath and getTestDataPath methods.
 */
abstract class PluginTestCase : BasePlatformTestCase() {

    override fun setUp() {
        mockkConstructor(AnalyticsStartupActivity::class)
        every { AnalyticsStartupActivity().runActivity(any()) } returns Unit

        super.setUp()

        val analyticsService = service<AnalyticsService>()
        analyticsService.isAnalyticsEnabled = true
        analyticsService.isUserAcknowledged = true
    }

    override fun getBasePath() = "/testData/"

    override fun getTestDataPath() = System.getProperty("user.dir") + basePath
}
```

Code 1: A child of the BasePlatformTestCase which mocks the license agreement. The class also overrides the getBasePath and getTestDataPath methods. The code can be found in For-Mainframe/src/apiTest/customTestCase/pluginTestCase.

final notes

You can run the API tests by going to the project root folder and running

./gradlew apiTest

after the project has been built.

```kotlin
/**
 * The function checking whether the sandboxCrudable and configCrudable are modified accrodingly
 */
fun assertCrudableConfig(connectionDialogStateList: List<ConnectionDialogState>) {
    var conConfigSet = emptySet<ConnectionConfig>()
    var creSet = emptyList<Credentials>()
    for (connectionDialogState in connectionDialogStateList) {
        conConfigSet += connectionDialogState.connectionConfig
        creSet += connectionDialogState.credentials
    }
    TestCase.assertEquals(conConfigSet.toList(),sandboxCrudable.getAll<ConnectionConfig>().toList())
    TestCase.assertEquals(conConfigSet.toList(),configCrudable.getAll<ConnectionConfig>().toList())
    TestCase.assertEquals(creSet,sandboxCrudable.getAll<Credentials>().toList())
}

/**
 * testing the onAdd method of ConnectionTableModel on API level,
 * meaning checking whether the sandboxCrudable and configCrudable are modified accrodingly
 */
fun testOnAddConfig() {
    conTab.onAdd(sandboxCrudable, conStateA)
    conTab.onAdd(sandboxCrudable, conStateB)
    conConfig.apply()
    assertCrudableConfig(listOf(conStateA, conStateB))
    conTab.onDelete(sandboxCrudable,conStateA)
    conTab.onDelete(sandboxCrudable,conStateB)
    conConfig.apply()
    assertCrudableConfig(listOf())
}
```

Code 2: Function checking whether the sandboxCrudable and configCrudable have been modified correctly along with a test for adding connection.