Unit tests

In this section, we will introduce the approach to unit test the For Mainframe plugin. When we want to unit test a class, there is a need to mock all the other components the class relies on. In plugin testing, that can be difficult. The IntelliJ says the following in their documentation.

"Another consequence of our testing approach is that we do not provide a recommended approach to mocking. We have a few tests in our codebase that use JMock. Still, in general, we find it difficult to mock all of the interactions with IntelliJ Platform components that your plugin class will need to have. We recommend working with real components instead."

However, in our case there are some components that can be unit tested. One of them is the ConnectionTableModel class, which is only reliant on the existence of an application. The IntelliJ provides the MockApplication class, which is enough for our purposes. We defined the UnitTestCase class, which sets up the Mock Application and can be seen in Code 1.

```kotlin
/**
 * A test case for unit tests, which only depend on the existence of an application.
 */
open class UnitTestCase {
    /**
     * spying on MockApplication
     */
    val app = spyk(MockApplication(Disposer.newDisposable( debugName: "")))

    /**
     * Setting up MockApplication
     */
    @BeforeEach
    fun setUp() {
        ApplicationManager.setApplication(app,Disposer.newDisposable( debugName: ""))
    }

    /**
     * Tearing down MockApplication
     */
    @AfterEach
    fun tearDown() {
        app.dispose()
    }
}
```

Code. 1: A test case for unit tests. The class testing the Connection Table Model will be a child of this test case. The code can be found here For-Mainframe/src/test/kotlin/eu.ibagroup.formainframe.cofig/UnitTestCase.

The class testing the ConnectionTableModel will be a child of the UnitTestCase, visible in

```kotlin
/**
 * Testing a class, which only needs the existence of an Application to function properly
 */
class ConnectionsTableModelTest: UnitTestCase() {
    val sandbox = ConfigSandboxImpl()
    val conTab = ConnectionsTableModel(sandbox.crudable)
    val connectionDialogStateA = ConnectionDialogState(connectionName = "a", connectionUrl = "https://a.com",
        username = "a", password = "a")
    val connectionDialogStateB = ConnectionDialogState(connectionName = "b", connectionUrl = "https://b.com",
        username = "b", password = "b")

    /**
     * tests the fetch method of ConnectionTableModel
     */
    @Test
    fun fetch() {
        conTab.addRow(connectionDialogStateA)
        assertEquals(mutableListOf(connectionDialogStateA),conTab.fetch(sandbox.crudable))
    }
```

Code. 2: The beginning of the class, which tests the Connection Table Model. It is a child of UnitTestCase. The code can be found in For-Mainframe/src/test/kotlin/eu.ibagroup.formainframe.cofig/connect/ui/ConnectionTableModelTest.

Code 2. Please ignore the shouldFail function in the code. We will return to it when we will be going over the GitHub Actions.

Also, not all functions are reliant on plugin components. One of these examples in the For Mainframe plugin is a function called hashCode in class CredentialsTest. Tests of this function can be seen in Code 3.

In conclusion, although mocking is not recommended, that does not mean the unit tests should be avoided. Methods that are not reliant on plugin components can be unit tested. Moreover since our example, the hashCode method, is tied to the security of user's data, it probably should be unit tested. Then there are classes that can be easily unit tested by setting up the MockApplication, like the ConnectionTableModel class. Since no complex mocking is required, there could be a value in unit testing these classes.

final notes

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

./gradlew test

after the project has been built or

./gradlew build

All of the tests within the test source set are automatically being run along with the build taks unless said otherwise.

```kotlin
/**
 * Test class, which does not need any plugin component to unit test
 */
class CredentialsTest {
    val connectionDialogState = ConnectionDialogState(connectionName = "a", connectionUrl = "https://a.com",
        username = "a", password = "a")

    /**
     * Tests the hashCode method of Credentials.
     */
    @Test
    fun testHashCode() {
        val credentials = Credentials(connectionDialogState.connectionConfig.uuid, connectionDialogState.username,
            connectionDialogState.password)
        val credentials2 = Credentials(connectionDialogState.connectionConfig.uuid, connectionDialogState.username,
            connectionDialogState.password)
        assertNotEquals(credentials.hashCode(),credentials2.hashCode())
    }
}
```

Code. 3: Testing a method called hashCode which does not need any plugin components. This test might need to be reworked depending on what requirements the hashCode has. For now it tests only non-equality of two hashCodes of credentials with the same values. The code can be found in For-Mainframe/src/test/kotlin/eu.ibagroup.formainframe.cofig/connect/CredentialsTest.