The Ultimate Guide to Android Testing (Unit Tests, UI Tests, End-to-End Tests) - Clean Architecture

https://www.youtube.com/watch?v=nDCCwyS0_MQ&t=14s&ab_channel=PhilippLackner

On big projects testing saves time because you can see with one click that after adding a feature you didn't break any existing feature.

Unit tests: tests single components (one class, one function, etc.).

Integration tests: tests two classes working together.

End-to-end tests: they simulate user behavior.

Test dependencies

```
    // Local unit tests
    testImplementation "androidx.test:core:1.4.0"
    testImplementation "junit:junit:4.13.2"
    testImplementation "androidx.arch.core:core-testing:2.1.0
    testImplementation "org.jetbrains.kotlinx:kotlinx-corouti
    testImplementation "com.google.truth:truth:1.1.3"
    testImplementation "com.squareup.okhttp3:mockwebserver:4.
    testImplementation "io.mockk:mockk:1.10.5"
    debugImplementation "androidx.compose.ui:ui-test-manifest

    // Instrumentation tests
    androidTestImplementation 'com.google.dagger:hilt-android
    kaptAndroidTest 'com.google.dagger:hilt-android-compiler:
    androidTestImplementation "junit:junit:4.13.2"
    androidTestImplementation "org.jetbrains.kotlinx:kotlinx-
    androidTestImplementation "androidx.arch.core:core-testin
```

```
    androidTestImplementation "com.google.truth:truth:1.1.3"
    androidTestImplementation 'androidx.test.ext:junit:1.1.3'
    androidTestImplementation 'androidx.test:core-ktx:1.4.0'
    androidTestImplementation "com.squareup.okhttp3:mockwebse
    androidTestImplementation "io.mockk:mockk-android:1.10.5"
    androidTestImplementation 'androidx.test:runner:1.4.0'
```

In your project you have a test directory and an android test directory. The test directory will contain your unit tests. The android test directory contains instrumented unit tests, they are related to Android, they need Android components, they don't run in the JVM, they need an emulator.

Writing our first unit test

In a Clean arquitecture we unit test the use cases and the utils functions.

Go to a class → Alt+enter → create test → select JUnit4 → select the test directory since we don't need Android components for this test.

```
package com.plcoding.cleanarchitecturenoteapp.feature_note.do

import com.google.common.truth.Truth.assertThat
import com.plcoding.cleanarchitecturenoteapp.feature_note.dat
import com.plcoding.cleanarchitecturenoteapp.feature_note.dat
import com.plcoding.cleanarchitecturenoteapp.feature_note.dom
import com.plcoding.cleanarchitecturenoteapp.feature_note.dom
import com.plcoding.cleanarchitecturenoteapp.feature_note.dom
import kotlinx.coroutines.flow.first
import kotlinx.coroutines.runBlocking
import org.junit.Before
import org.junit.Test

class GetNotesTest {

    private lateinit var getNotes: GetNotes
    private lateinit var fakeRepository: FakeNoteRepository

    @Before
```

```
    fun setUp() {
        fakeRepository = FakeNoteRepository()
        getNotes = GetNotes(fakeRepository)

        val notesToInsert = mutableListOf<Note>()
        ('a'..'z').forEachIndexed { index, c ->
            notesToInsert.add(
                Note(
                    title = c.toString(),
                    content = c.toString(),
                    timestamp = index.toLong(),
                    color = index
                )
            )
        }
        notesToInsert.shuffle()
        runBlocking {
            notesToInsert.forEach { fakeRepository.insertNote
        }
    }

    @Test
    fun `Order notes by title ascending, correct order`() = r
        val notes = getNotes(NoteOrder.Title(OrderType.Ascend

        for(i in 0..notes.size - 2) {
            assertThat(notes[i].title).isLessThan(notes[i+1].
        }
    }

    @Test
    fun `Order notes by title descending, correct order`() =
        val notes = getNotes(NoteOrder.Title(OrderType.Descen

        for(i in 0..notes.size - 2) {
            assertThat(notes[i].title).isGreaterThan(notes[i+
        }
    }
```

```
    @Test
    fun `Order notes by date ascending, correct order`() = ru
        val notes = getNotes(NoteOrder.Date(OrderType.Ascendi

        for(i in 0..notes.size - 2) {
            assertThat(notes[i].timestamp).isLessThan(notes[i
        }
    }

    @Test
    fun `Order notes by date descending, correct order`() = r
        val notes = getNotes(NoteOrder.Date(OrderType.Descend

        for(i in 0..notes.size - 2) {
            assertThat(notes[i].timestamp).isGreaterThan(note
        }
    }

    @Test
    fun `Order notes by color ascending, correct order`() = r
        val notes = getNotes(NoteOrder.Color(OrderType.Ascend

        for(i in 0..notes.size - 2) {
            assertThat(notes[i].color).isLessThan(notes[i+1].
        }
    }

    @Test
    fun `Order notes by color descending, correct order`() =
        val notes = getNotes(NoteOrder.Color(OrderType.Descen

        for(i in 0..notes.size - 2) {
            assertThat(notes[i].color).isGreaterThan(notes[i+
        }
    }
}
```

The setUp() function will run before every unit test.

If you need a repository or some dependency you create a fake one. In the example shown before there is a fake repository. This is the code for that:

```
package com.plcoding.cleanarchitecturenoteapp.feature_note.da

import com.plcoding.cleanarchitecturenoteapp.feature_note.dom
import com.plcoding.cleanarchitecturenoteapp.feature_note.dom
import kotlinx.coroutines.flow.Flow
import kotlinx.coroutines.flow.flow

class FakeNoteRepository : NoteRepository {

    private val notes = mutableListOf<Note>()

    override fun getNotes(): Flow<List<Note>> {
        return flow { emit(notes) }
    }

    override suspend fun getNoteById(id: Int): Note? {
        return notes.find { it.id == id }
    }

    override suspend fun insertNote(note: Note) {
        notes.add(note)
    }

    override suspend fun deleteNote(note: Note) {
        notes.remove(note)
    }
}
```

In the setUp function we can also add the necessary data for the tests.

@Test annotation is to tell that a function is a test.

The assertThat function we are using is from the truth dependency.

Now practice: check out the code at https://github.com/philipplackner/CleanArchitectureNoteApp/tree/testing and test the AddNote.kt:

- A blank title note should return the InvalidNoteException.
- A blank content note should return the InvalidNoteException.
- Otherwise the note should be added.

Integration UI tests

We will test that the notes screen when toggling the sort button it displays the sorting options.

Alt+enter on NotesScreen function signature → create test → select the androidTest folder.

If you are ussing Dagger-Hilt you need to annotate the test with @HiltAndroidTest and create the test module.
@UninstallModules(AppModule::class) is used to don't use the real module.

Dagger-Hilt also needs a class annotated with . That's why we need to create the HiltTestRunner class:

```
package com.plcoding.cleanarchitecturenoteapp

import android.app.Application
import android.content.Context
import androidx.test.runner.AndroidJUnitRunner
import dagger.hilt.android.testing.HiltTestApplication

class HiltTestRunner : AndroidJUnitRunner() {

    override fun newApplication(cl: ClassLoader?, className: 
        return super.newApplication(cl, HiltTestApplication::
    }
}
```

After that you need to go to build.graddle and change the testInstrumentationRunner.

We also need to define a Rule. This will help us with some behavior like intents or inject hilt dependencies:

```
@get:Rule(order = 0)
val hiltRule = HiltAndroidRule(this)

@get:Rule(order = 1)
val composeRule = createAndroidComposeRule<MainActivity>()
```

This is the code for the integration test explained before:

```
package com.plcoding.cleanarchitecturenoteapp.feature_note.pr

import android.content.Context
import androidx.compose.animation.ExperimentalAnimationApi
import androidx.compose.ui.test.assertIsDisplayed
import androidx.compose.ui.test.junit4.createAndroidComposeRu
import androidx.compose.ui.test.onNodeWithContentDescription
import androidx.compose.ui.test.onNodeWithTag
import androidx.compose.ui.test.performClick
import androidx.navigation.compose.NavHost
import androidx.navigation.compose.composable
import androidx.navigation.compose.rememberNavController
import androidx.test.core.app.ApplicationProvider
import com.plcoding.cleanarchitecturenoteapp.core.util.TestTa
import com.plcoding.cleanarchitecturenoteapp.di.AppModule
import com.plcoding.cleanarchitecturenoteapp.feature_note.pre
import com.plcoding.cleanarchitecturenoteapp.feature_note.pre
import com.plcoding.cleanarchitecturenoteapp.ui.theme.CleanAr
import dagger.hilt.android.testing.HiltAndroidRule
import dagger.hilt.android.testing.HiltAndroidTest
import dagger.hilt.android.testing.UninstallModules
import org.junit.Assert.*
import org.junit.Before
import org.junit.Rule
import org.junit.Test
```

```
@HiltAndroidTest
@UninstallModules(AppModule::class)
class NotesScreenTest {

    @get:Rule(order = 0)
    val hiltRule = HiltAndroidRule(this)

    @get:Rule(order = 1)
    val composeRule = createAndroidComposeRule<MainActivity>(

    @ExperimentalAnimationApi
    @Before
    fun setUp() {
        hiltRule.inject()
        composeRule.setContent {
            val navController = rememberNavController()
            CleanArchitectureNoteAppTheme {
                NavHost(
                    navController = navController,
                    startDestination = Screen.NotesScreen.rou
                ) {
                    composable(route = Screen.NotesScreen.rou
                        NotesScreen(navController = navContro
                    }
                }
            }
        }
    }

    @Test
    fun clickToggleOrderSection_isVisible() {
        composeRule.onNodeWithTag(TestTags.ORDER_SECTION).ass
        composeRule.onNodeWithContentDescription("Sort").perf
        composeRule.onNodeWithTag(TestTags.ORDER_SECTION).ass
    }
}
```

The testTags need to be defined in the component you are searching.

The “Sort” string used should be a reference to R.string in case it is changed in the future. For getting the context in the test we use: val context = ApplicationProvider.getApplicationContext<Context>()

End to end tests

This will simulate what the user does in an app.

We will have the same test setup as the example done before.

We will test that we can add a new note and edit it:

```
package com.plcoding.cleanarchitecturenoteapp.feature_note.pr

import androidx.compose.animation.ExperimentalAnimationApi
import androidx.compose.ui.test.*
import androidx.compose.ui.test.junit4.createAndroidComposeRu
import androidx.navigation.NavType
import androidx.navigation.compose.NavHost
import androidx.navigation.compose.composable
import androidx.navigation.compose.rememberNavController
import androidx.navigation.navArgument
import com.plcoding.cleanarchitecturenoteapp.core.util.TestTa
import com.plcoding.cleanarchitecturenoteapp.di.AppModule
import com.plcoding.cleanarchitecturenoteapp.feature_note.pre
import com.plcoding.cleanarchitecturenoteapp.feature_note.pre
import com.plcoding.cleanarchitecturenoteapp.feature_note.pre
import com.plcoding.cleanarchitecturenoteapp.ui.theme.CleanAr
import dagger.hilt.android.testing.HiltAndroidRule
import dagger.hilt.android.testing.HiltAndroidTest
import dagger.hilt.android.testing.UninstallModules
import org.junit.Before
import org.junit.Rule
import org.junit.Test

@HiltAndroidTest
@UninstallModules(AppModule::class)
class NotesEndToEndTest {

    @get:Rule(order = 0)
```

```
    val hiltRule = HiltAndroidRule(this)

    @get:Rule(order = 1)
    val composeRule = createAndroidComposeRule<MainActivity>(

    @ExperimentalAnimationApi
    @Before
    fun setUp() {
        hiltRule.inject()
        composeRule.setContent {
            CleanArchitectureNoteAppTheme {
                val navController = rememberNavController()
                NavHost(
                    navController = navController,
                    startDestination = Screen.NotesScreen.rou
                ) {
                    composable(route = Screen.NotesScreen.rou
                        NotesScreen(navController = navContro
                    }
                    composable(
                        route = Screen.AddEditNoteScreen.rout
                                "?noteId={noteId}&noteColor={
                        arguments = listOf(
                            navArgument(
                                name = "noteId"
                            ) {
                                type = NavType.IntType
                                defaultValue = -1
                            },
                            navArgument(
                                name = "noteColor"
                            ) {
                                type = NavType.IntType
                                defaultValue = -1
                            },
                        )
                    ) {
                        val color = it.arguments?.getInt("not
```

```
                        AddEditNoteScreen(
                            navController = navController,
                            noteColor = color
                        )
                    }
                }
            }
        }
    }

    @Test
    fun saveNewNote_editAfterwards() {
        // Click on FAB to get to add note screen
        composeRule.onNodeWithContentDescription("Add").perfo

        // Enter texts in title and content text fields
        composeRule
            .onNodeWithTag(TestTags.TITLE_TEXT_FIELD)
            .performTextInput("test-title")
        composeRule
            .onNodeWithTag(TestTags.CONTENT_TEXT_FIELD)
            .performTextInput("test-content")
        // Save the new
        composeRule.onNodeWithContentDescription("Save").perf

        // Make sure there is a note in the list with our tit
        composeRule.onNodeWithText("test-title").assertIsDisp
        // Click on note to edit it
        composeRule.onNodeWithText("test-title").performClick

        // Make sure title and content text fields contain no
        composeRule
            .onNodeWithTag(TestTags.TITLE_TEXT_FIELD)
            .assertTextEquals("test-title")
        composeRule
            .onNodeWithTag(TestTags.CONTENT_TEXT_FIELD)
            .assertTextEquals("test-content")
        // Add the text "2" to the title text field
```

```
        composeRule
            .onNodeWithTag(TestTags.TITLE_TEXT_FIELD)
            .performTextInput("2")
        // Update the note
        composeRule.onNodeWithContentDescription("Save").perf

        // Make sure the update was applied to the list
        composeRule.onNodeWithText("test-title2").assertIsDis
    }

    @Test
    fun saveNewNotes_orderByTitleDescending() {
        for(i in 1..3) {
            // Click on FAB to get to add note screen
            composeRule.onNodeWithContentDescription("Add").p

            // Enter texts in title and content text fields
            composeRule
                .onNodeWithTag(TestTags.TITLE_TEXT_FIELD)
                .performTextInput(i.toString())
            composeRule
                .onNodeWithTag(TestTags.CONTENT_TEXT_FIELD)
                .performTextInput(i.toString())
            // Save the new
            composeRule.onNodeWithContentDescription("Save").
        }

        composeRule.onNodeWithText("1").assertIsDisplayed()
        composeRule.onNodeWithText("2").assertIsDisplayed()
        composeRule.onNodeWithText("3").assertIsDisplayed()

        composeRule
            .onNodeWithContentDescription("Sort")
            .performClick()
        composeRule
            .onNodeWithContentDescription("Title")
            .performClick()
        composeRule
```

```
            .onNodeWithContentDescription("Descending")
            .performClick()

        composeRule.onAllNodesWithTag(TestTags.NOTE_ITEM)[0]
            .assertTextContains("3")
        composeRule.onAllNodesWithTag(TestTags.NOTE_ITEM)[1]
            .assertTextContains("2")
        composeRule.onAllNodesWithTag(TestTags.NOTE_ITEM)[2]
            .assertTextContains("1")
    }
}
```

The first end to end test adds a new note, checks that it exists, edits it and checks that it is edited.

The second end to end test checks that 3 notes are created and them are sorted correctly.

If you want to run all instrumented tests go to folder the folder view, right click on ‘com (androidTest)’ and click on “Run tests in ‘com’”.