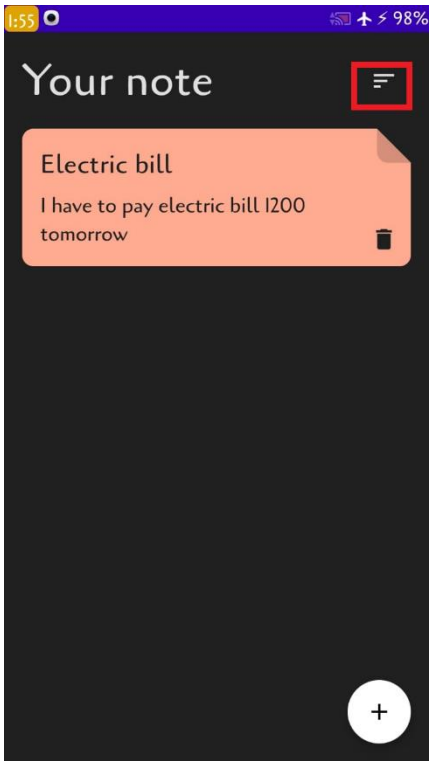
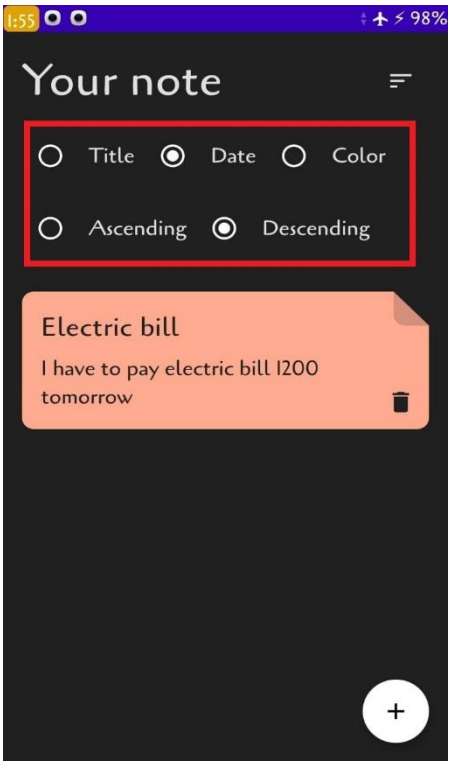
UI testing of a Composable Screen

What is UI/Instrumented testing?

Using android API/framework(Activity, Fragment, Context, etc) when we perform any action on the particular screen and against this, we are expecting the intended result we can say we are doing UI testing. In this type of testing, we need a physical device or emulator.

In this tutorial, our goal is to test the visibility of the order section of a Note App. When we will click the sort icon order section should appear.





Set-Up:

Sadly, the Initial setup is painful to run UI tests. But once you set it up it is very easy to run our test. It is important to see that in this case, the project I'm showing is using hilt as dependency injection.

If you are using something else probably you'll need to change or add some other dependencies in order to set up your UI tests. After the setup part, everything remains the same.

Let's start by adding the following dependencies to our App build. Gradle file:

```
// Instrumentation tests
    androidTestImplementation 'com.google.dagger:hilt-android-testing:2.37'
    kaptAndroidTest 'com.google.dagger:hilt-android-compiler:2.43'
    androidTestImplementation "junit:junit:4.13.2"
    androidTestImplementation "org.jetbrains.kotlinx:kotlinx-coroutines-test:1.6.4"
    androidTestImplementation "androidx.arch.core:core-testing:2.1.0"
    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 'androidx.test:runner:1.4.0'
    androidTestImplementation 'androidx.compose.ui:ui-test-junit4:1.3.0-alpha02'
```

N.B: please use the latest version of all libraries

As we are using Hilt here, to work it properly it needs an Application class, to make this happen we have to create a HiltTestRunner class in the root folder of the AndroidTest package.

```
class HiltTestRunner : AndroidJUnitRunner() {
 override fun newApplication(
        cl: ClassLoader?,
        className: String?,
        context: Context?
    ): Application {
        return super.newApplication(cl, HiltTestApplication::class.java.name, context)
    }
}
```

After that, we have to go to our App build.Gradle file in between android block there will be called test instrumentation runner, here we have to provide our full path name like bellow:

testInstrumentationRunner"com.purnendu.testingwithandroid.HiltTestRunne"

It is always good practice not to use production AppModule for Hilt, we will create a separate one in the root folder of the AndroidTest package for testing only.

```
@Module
@InstallIn(SingletonComponent::class)
object TestAppModule
{
  @Provides
   @Singleton
   fun provideNoteDatabase( app: Application): NoteDatabase {
      return Room.inMemoryDatabaseBuilder(
         app,
         NoteDatabase::class.java,
      ).build()
   }
  @Provides
   @Singleton
   fun provideNoteRepository(db: NoteDatabase): NoteRepository {
      return NoteRepositoryImpl(db.noteDao)
   }
  @Provides
   @Singleton
   fun provideNoteUseCases(repository: NoteRepository): NoteUseCases {
      return NoteUseCases(
         getNotes = GetNotes(repository),
         deleteNote = DeleteNote(repository),
         addNote = AddNote(repository),
         getNote = GetNote(repository)
      )
   }
}
```

N.B: Notices we are using inMemoryDatabaseBuilder for creating a database in memory while testing so that it will only persist run time only.

Now let's begin the actual coding part:

Coding:

Create a class named NotesScreenTest(for my project)

To avoid crashing add the following function outside the class:

```
private fun AndroidComposeTestRule<ActivityScenarioRule<MainActivity>,
MainActivity>.clearAndSetContent(content: @Composable () -> Unit) {

    (this.activity.findViewById<ViewGroup>(android.R.id.content)?.getChildAt(0) as?
ComposeView)?.setContent(content)

        ?: this.setContent(content)

  }
```

Annotate the class with the following annotation

@HiltAndroidTest

@UninstallModules(AppModule::class)

The first annotation for Hilt and the second one is to avoid conflict on between AppModule and TestAppModule.

Now add a rule by applying for an order in class like

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

   val hiltRule = HiltAndroidRule(this)

 @get:Rule(order = 1)

   val composeRule = createAndroidComposeRule<MainActivity>()
```

The first one is for Hilt the and second rule is for Compose by mentioning the activity we want to test.

Next, we have to set up a function that will run before every test case, So creating a function named setup and annotation it @before like

```
@ExperimentalAnimationApi
  @Before
  fun setUp() {
    hiltRule.inject()
    composeRule.clearAndSetContent {
      val navController = rememberNavController()
      TestingWithAndroidTheme {
        NavHost(
          navController = navController,
          startDestination = Screen.NotesScreen.route
        ) {
          composable(route = Screen.NotesScreen.route) {
            NotesScreen(navController = navController)
          }
        }
      }
    }
  }
```

In this function, first of all, we inject the Hilt rule, and then using compose rule we add our intended test screen in the Activity.

Now we can write our intended test ->

So, first of all, we create a test function named clickToggleOrderSection_isVisible() by annotating it like @Test

In this function, we have to write our test.

In the beginning, we have told that initially our order section of the NoteApp should not be displayed after performing a click to Sort icon this should be displayed.

So, be sure to add a tag or content description of the intended action element of composable like in my case:

```
OrderSection(
        modifier = Modifier
            .testTag("ORDER_SECTION"),
    )
Or,
Icon( imageVector = Icons.Default.Sort, contentDescription = "Sort" )
```

After that, we have to assert that the order section is not visible by writing

composeRule.onNodeWithTag("ORDER_SECTION").assertDoesNotExist()

Next, we have to perform click on the Sort icon

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

At last, we have to ensure that now order section is visible by writing

composeRule.onNodeWithTag("ORDER_SECTION").assertIsDisplayed()

```
@Test
fun clickToggleOrderSection_isVisible() {
    composeRule.onNodeWithTag("ORDER_SECTION").assertDoesNotExist()
    composeRule.onNodeWithContentDescription("Sort").performClick()
    composeRule.onNodeWithTag("ORDER_SECTION").assertIsDisplayed()
}
```

If everything is all right we can run our test by clicking the green arrow besides our

Testing function, but be sure you are connected with a physical device or emulator.

Conclusion:

It wasn't so hard to test and validate if our screen is working as we expect. With the help of unit & instrumented tests running successfully, we can be sure that our app is going to have fewer or no side effects/bugs.