11

Android Architecture Components

Activity 11.01 – shopping notes app

Solution

Perform the following steps to solve the problem:

1. Create a new Android Studio project with an empty activity.
2. Add the `ksp` plugin dependency to `gradle/libs.versions.toml`:

```
[versions]
…
ksp = "2.0.21-1.0.25"
…
[plugins]
…
ksp = { id = "com.google.devtools.ksp", version.ref = "ksp" }
```

3. In the root `build.gradle.kts` file, add the `ksp` plugin:

```
plugins {
    …
    alias(libs.plugins.ksp) apply false
}
```

4. In `app/build.gradle.kts`, add the `ksp` plugin:

```
plugins {
    …
    alias(libs.plugins.ksp)

}
```

5. In `gradle/libs.versions.toml`, add the required dependencies:

```
[versions]
…
room = "2.7.1"
viewModelCompose = "2.8.7"

[libraries]
…
room-runtime = { group = "androidx.room", name = "room-runtime",
version.ref = "room" }
room-compiler = { group = "androidx.room", name = "room-compiler",
version.ref = "room" }
room-ktx = { group = "androidx.room", name = "room-ktx", version.ref
= "room" }
androidx-viewmodel-compose = { group = "androidx.lifecycle", name =
"lifecycle-viewmodel-compose", version.ref = "viewModelCompose" }
…
```

6. Next, let's add the libraries in `app/build.gradle.kts`:

```
    implementation(libs.androidx.viewmodel.compose)
    implementation(libs.room.runtime)
    implementation(libs.room.ktx)
    ksp(libs.room.compiler)
```

7. Create a class called `Note`, which will be a Room `@Entity` annotation:

```
@Entity(tableName = "notes")
data class Note(
    @PrimaryKey(autoGenerate = true)
    @ColumnInfo(name = "id") val id: Long = 0,
    @ColumnInfo(name = "text") val text: String
)
```

8. Create a NoteDao class, which will manage Note entities:

```
@Dao
interface NoteDao {

    @Insert(onConflict = OnConflictStrategy.REPLACE)
    suspend fun insertNote(note: Note)

    @Query("SELECT * FROM notes")
    fun loadNotes(): Flow<List<Note>>

    @Query("SELECT count(*) FROM notes")
    fun loadNoteCount(): Flow<Int>
}
```

9. Now, add the Note entity to NotesDatabase, as follows:

```
@Database(
    entities = [Note::class],
    version = 1
)
abstract class NotesDatabase : RoomDatabase() {

    abstract fun noteDao(): NoteDao
}
```

10. Now, let's define a repository. The Repository pattern is useful in situations where you have one or more sources of data (server, room, or memory) that can be combined, modified, and processed. This will help us centralize our access to the data and decouple the application code from the data sources. In our case, the only data source we have is Room, so our repository will act as a wrapper over NoteDao:

```
interface NoteRepository {

    suspend fun insertNote(note: Note)

    fun getAllNotes(): Flow<List<Note>>

    fun getNoteCount(): Flow<Int>
}
```

11. Now, let's add the implementation of our repository:

```
class NoteRepositoryImpl(
    private val noteDao: NoteDao
) : NoteRepository {
    override suspend fun insertNote(note: Note) =
        noteDao.insertNote(note)

    override fun getAllNotes(): Flow<List<Note>> =
        noteDao.loadNotes()

    override fun getNoteCount(): Flow<Int> =
        noteDao.loadNoteCount()
}
```

12. Let's create a `NoteViewModel` class with the `UiState` class defined inside of it:

```
class NoteViewModel(
    private val noteRepository: NoteRepository
) : ViewModel() {
    data class UiState(
        val notes: List<String> = emptyList(),
        val noteCount: Int = 0
    )
}
```

13. Let's create a `NoteViewModel` class. This will take the notes from `NoteRepository` and convert them into `UiState` objects, which will hold each note's information and the total note count:

```
class NoteViewModel(
    private val noteRepository: NoteRepository
) : ViewModel() {

    private val _state = MutableStateFlow(UiState())
    val state: StateFlow<UiState> = _state

    init {
        viewModelScope.launch {
            noteRepository.getAllNotes()
```

```
                  .combine(
                      noteRepository.getNoteCount()
                  ) { notes, count ->
                  UiState(
                      notes = notes.map { it.text },
                      noteCount = count
                  )
                  }
                  .collectLatest {
                      _state.emit(it)
                  }
          }
      }

      fun insertNote(text: String) {
          viewModelScope.launch {
              noteRepository.insertNote(
                  Note(id = 0, text = text)
              )
          }
      }
  ...
  }
```

14. In `NoteViewModel`, add a function that will insert a new note into the `NoteRespository` object:

```
class NoteViewModel(
    private val noteRepository: NoteRepository
) : ViewModel() {

    ...
    fun insertNote(text: String) {
        viewModelScope.launch {
            noteRepository.insertNote(
                Note(id = 0, text = text)
            )
        }
    }
...
}
```

15. In `app/res/values/strings.xml`, add the strings we will need to display to the user:

```
    <string name="click_me">Click Me</string>
    <string name="total_notes">Total: %d</string>
```

16. In `MainActivity`, create a `@Composable NoteScreen` function, which will show a `TextField` element and a `Button` element for adding new notes, then a `Text` element showing the total note count, and finally the list of notes:

```
@Composable
fun NoteScreen(
    uiState: NoteViewModel.UiState,
    onNewNoteClicked: (String) -> Unit,
    modifier: Modifier = Modifier
) {
    LazyColumn(
        modifier = modifier.fillMaxSize(),
        verticalArrangement = Arrangement.Center,
        horizontalAlignment = Alignment.CenterHorizontally
    ) {
        item {
            var textFieldText by remember {
                mutableStateOf("")
            }
            TextField(
                value = textFieldText,
                onValueChange = {
                    textFieldText = it
                }
            )
            Button(onClick = {
                onNewNoteClicked(textFieldText)
            }) {
                Text(
                    text = stringResource(
                        id = R.string.click_me)
                )
            }
            Text(
                text = stringResource(
                    id = R.string.total_notes,
                    uiState.noteCount
                )
            )
        }
        items(uiState.notes.size) {
            Text(text = uiState.notes[it])
        }

    }
}
```

17. In `MainActivity`, create a `@Composable Note` function. This will connect the `NoteViewModel` object, which will be used to set the state in `NoteScreen`:

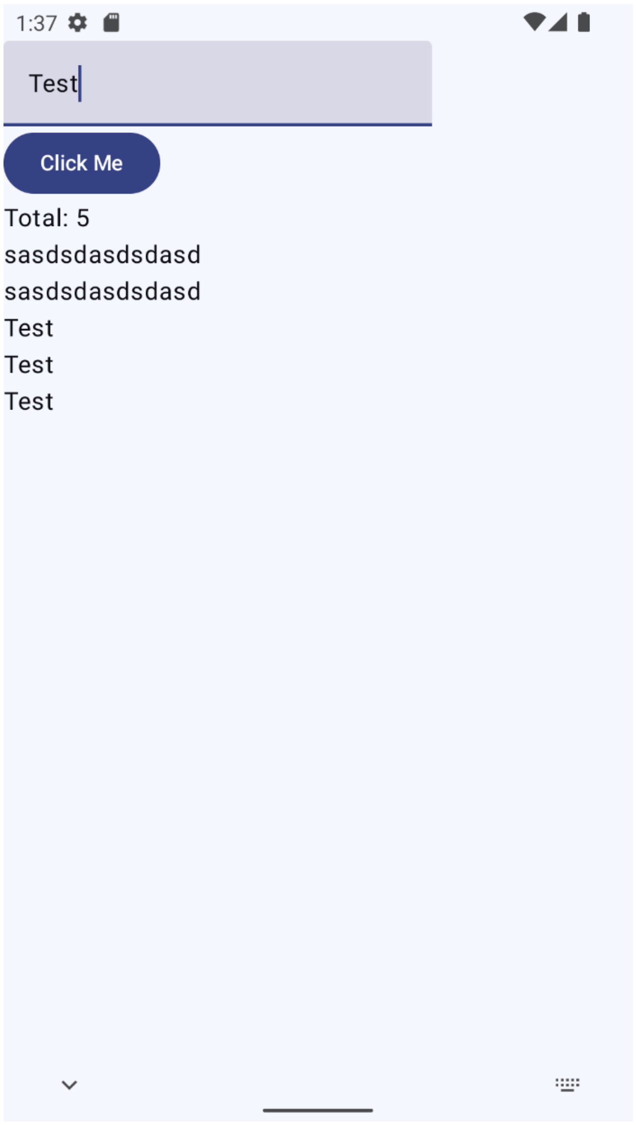
```
@Composable
fun Note(
    viewModel: NoteViewModel,
    modifier: Modifier = Modifier
) {
    NoteScreen(
        uiState =
            viewModel.state.collectAsState().value,
        onNewNoteClicked = {
            viewModel.insertNote(it)
        }, modifier = modifier
    )
}
```

18. Modify the `MainActivity onCreate` method to create the `NoteDatabase` object, create the `NoteRepository` object, and then inject the `NoteRepository` object into the `NoteViewModel` object. Then, invoke the `Note` function created in the previous step:

```
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    enableEdgeToEdge()
```

```
        val noteDatabase = Room.databaseBuilder(
            applicationContext,
            NoteDatabase::class.java,
            "notes-db"
        ).build()
        val noteRepository =
            NoteRepositoryImpl(noteDatabase.noteDao())
        setContent {
            Activity1101Theme {
                Scaffold(
                    modifier = Modifier.fillMaxSize()
                ) { innerPadding ->
                    val viewModel =
                        viewModel<NoteViewModel>(
                            factory = object :
                            ViewModelProvider.Factory {
                                override fun
                                    <T : ViewModel>
                                        create(
                                            modelClass:
                                            Class<T>
                                        ): T {
                                            return
                                            NoteViewModel(
                                            noteRepository
                                            ) as T
                                        }
                            }
                        )
                    Note(
                        viewModel,
                        Modifier.padding(innerPadding)
                    )
                }
            }
        }
    }
```

19. If you run the application, it should look like the following screenshot:



Total: 5
sasdsdasdsdasd
sasdsdasdsdasd
Test
Test
Test

Figure 11.1 – Output of Activity 11.01