Unit 6 — Data Persistence (Room, DataStore, Files)

Mobile Application Development — Lecture 6

Masoud Hamad

State University of Zanzibar (SUZA)

Semester II, 2025/2026

Outline

Storage Options on Android

Option	**Use for**
DataStore (Preferences)	small key–value settings
DataStore (Proto)	typed structured settings
Room (SQLite)	structured relational data
Files	images, cached blobs, docs
Retrofit + REST	server-backed data

Dependencies

```
// build.gradle.kts
implementation("androidx.room:room-runtime:2.6.1")
implementation("androidx.room:room-ktx:2.6.1")
ksp("androidx.room:room-compiler:2.6.1")
```

Entity (Table)

```
@Entity(tableName = "students")
data class Student(
    @PrimaryKey(autoGenerate = true) val id: Int = 0,
    val name: String,
    val course: String,
    val year: Int
)
```

Each `@Entity` maps to a SQLite table; each property to a column.

DAO (Data Access Object)

```
@Dao
interface StudentDao {
    @Insert suspend fun insert(s: Student): Long
    @Update suspend fun update(s: Student)
    @Delete suspend fun delete(s: Student)

    @Query("SELECT * FROM students ORDER BY name ASC")
    fun getAll(): Flow<List<Student>>

    @Query("SELECT * FROM students WHERE id = :id")
    suspend fun getById(id: Int): Student?
}
```

Database

```
@Database(entities = [Student::class], version = 1)
abstract class AppDatabase : RoomDatabase() {
    abstract fun studentDao(): StudentDao

    companion object {
        @Volatile private var INSTANCE: AppDatabase? = null
        fun getDatabase(context: Context): AppDatabase =
            INSTANCE ?: synchronized(this) {
                Room.databaseBuilder(
                    context, AppDatabase::class.java, "app_db"
                ).build().also { INSTANCE = it }
            }
    }
}
```

Repository

```
class StudentRepository(private val dao: StudentDao) {
    val students: Flow<List<Student>> = dao.getAll()
    suspend fun add(student: Student) = dao.insert(student)
    suspend fun remove(student: Student) = dao.delete(student)
}
```

Repository hides the DAO (or API) from the ViewModel.

ViewModel with Room

```
class StudentViewModel(private val repo: StudentRepository) : ViewModel() {
    val students = repo.students
        .stateIn(viewModelScope,
                 SharingStarted.WhileSubscribed(5000),
                 emptyList())

    fun addStudent(name: String, course: String, year: Int) =
        viewModelScope.launch {
            repo.add(Student(name = name, course = course, year = year))
        }
}
```

Compose UI Binding

```
@Composable
fun StudentListScreen(viewModel: StudentViewModel) {
    val students by viewModel.students.collectAsState()

    LazyColumn {
        items(students, key = { it.id }) { s ->
            Text("${s.name} -- ${s.course}")
        }
    }
}
```

Flow & Coroutines

- `suspend fun` — asynchronous, cannot block the main thread
- `Flow<T>` — cold async stream of values
- `viewModelScope.launch { }` — coroutine tied to ViewModel lifecycle
- In Composables: `.collectAsState()` converts Flow → State

Main thread rule

Never do disk or network work on the main thread. Always mark it `suspend` or expose as `Flow`.

DataStore (Preferences)

```
val Context.dataStore by preferencesDataStore(name = "settings")

class SettingsRepository(private val context: Context) {
    private val DARK_MODE = booleanPreferencesKey("dark_mode")

    val darkMode: Flow<Boolean> = context.dataStore.data
        .map { it[DARK_MODE] ?: false }

    suspend fun setDarkMode(enabled: Boolean) {
        context.dataStore.edit { it[DARK_MODE] = enabled }
    }
}
```

File Storage

- **Internal** (app-private): `context.filesDir`
- **Cache** (transient): `context.cacheDir`
- **External** (shared/public): requires permissions

```
val file = File(context.filesDir, "notes.txt")
file.writeText("Hello SUZA")
val content = file.readText()
```

Migrations

```
Room.databaseBuilder(context, AppDatabase::class.java, "app_db")
    .addMigrations(MIGRATION_1_2)
    .build()
```

For prototypes: `.fallbackToDestructiveMigration()` (wipes old data — never do this in production).

Summary

- Room = recommended SQLite ORM; built on coroutines and Flow
- Standard layered design: Entity + DAO + Database + Repository + ViewModel
- DataStore for small settings, Files for blobs
- Always use `suspend` / `Flow` to stay off the main thread

Next: Networking with Retrofit and REST APIs.

Questions?