CS 426 – Mobile Application Development

Week 9: Lab – Galaxy Explorer with Persistence (Room + SharedPreferences)
Instructor: Dr. Slim Namouchi
Date: 15 Nov 2025

Lab Introduction

In Week 8, you built the Galaxy Explorer app, a dynamic Android application that displayed a scrollable list of galaxies using RecyclerView and navigated to detail screens with fragment-based navigation. While the app successfully fetched data from a static list and demonstrated the basics of MVVM separation, it lacked data persistence, meaning any data or changes would be lost when the app was closed or restarted.

In professional mobile development, data persistence is a core requirement. Users expect apps to remember their data, preferences, and state across sessions. Android provides robust solutions for local data storage:

- Room Database – a lightweight abstraction over SQLite that enables structured data storage with compile-time checks, LiveData, and coroutine support.
- SharedPreferences – a simple key-value storage mechanism, ideal for small user preferences or login states.

This lab extends your Week 8 project to integrate these persistence mechanisms. By the end of the lab, your app will:

1. Persist galaxy data locally using Room, allowing the app to display galaxy information even offline.
2. Observe and update data in real time using LiveData and ViewModel architecture.
3. Maintain user login state using SharedPreferences, ensuring a smooth user experience across app launches.
4. Introduce you to professional Android patterns such as repository abstraction, database initialization, and lifecycle-aware components.

Pre-requisites

- Completed Week 8 lab.
- Knowledge of RecyclerView, Fragments, Glide, Safe Args.

Deliverable: GitHub repo or ZIP containing Android Studio project, plus screenshots **or** a short video of the app running.

Step 1: Add dependencies

Do it as usual.

Step 2: Create GalaxyEntity

```
@Entity(tableName = "galaxies")
data class GalaxyEntity(
    @PrimaryKey val id: Int,
    val name: String,
    val subtitle: String,
    val description: String,
    val imageUrl: String
)
```

Step 3: Create DAO

```
@Dao
interface GalaxyDao {
    @Query("SELECT * FROM galaxies")
    fun getAllGalaxies(): Flow<List<GalaxyEntity>>

    @Query("SELECT * FROM galaxies WHERE id = :id")
    suspend fun getGalaxyById(id: Int): GalaxyEntity?

    @Insert(onConflict = OnConflictStrategy.REPLACE)
    suspend fun insertAll(galaxies: List<GalaxyEntity>)
}
```

Step 4: Create Room Database

```
@Database(entities = [GalaxyEntity::class], version = 1)
abstract class AppDatabase : RoomDatabase() {
    abstract fun galaxyDao(): GalaxyDao

    companion object {
        @Volatile private var instance: AppDatabase? = null

        fun getDatabase(context: Context): AppDatabase =
            instance ?: synchronized(this) {
                instance ?: Room.databaseBuilder(
                    context.applicationContext,
                    AppDatabase::class.java,
                    "galaxy_db"
                ).build().also { instance = it }
            }
    }
}
```

Step 5: Create Repository

```
class GalaxyRepository(private val dao: GalaxyDao) {
    val galaxies: Flow<List<GalaxyEntity>> = dao.getAllGalaxies()
```

```
    suspend fun insertGalaxies(galaxies: List<GalaxyEntity>) =
dao.insertAll(galaxies)

    suspend fun getGalaxyById(id: Int) = dao.getGalaxyById(id)
}
```

Step 6: Create ViewModel

```
class GalaxyViewModel(application: Application) : AndroidViewModel(application)
{
    private val repo: GalaxyRepository
    val galaxies: LiveData<List<GalaxyEntity>>

    init {
        val dao = AppDatabase.getDatabase(application).galaxyDao()
        repo = GalaxyRepository(dao)
        galaxies = repo.galaxies.asLiveData()
    }

    fun populateInitialData() = viewModelScope.launch {
        if (galaxies.value.isNullOrEmpty()) {
            val sample = SampleGalaxies.GALAXIES.map {
                GalaxyEntity(it.id, it.name, it.subtitle, it.description,
it.imageUrl)
            }
            repo.insertGalaxies(sample)
        }
    }

    suspend fun getGalaxyById(id: Int) = repo.getGalaxyById(id)
}
```

Step 7: Update List Fragment

```
class GalaxyListFragment : Fragment() {
    private val viewModel: GalaxyViewModel by viewModels()
    private lateinit var adapter: GalaxyAdapter

    override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
        adapter = GalaxyAdapter(emptyList()) { galaxy ->
            val action =
GalaxyListFragmentDirections.actionListToDetail(galaxy.id)
            findNavController().navigate(action)
        }
        binding.rvGalaxies.adapter = adapter
        binding.rvGalaxies.setHasFixedSize(true)

        viewModel.galaxies.observe(viewLifecycleOwner) { list ->
            adapter.updateList(list)
        }

        viewModel.populateInitialData()
    }
}
```

Add method in adapter:

```
fun updateList(newList: List<GalaxyEntity>) {
    items = newList
    notifyDataSetChanged()
}
```

Step 8: Update Detail Fragment

```
class GalaxyDetailFragment : Fragment() {
    private val viewModel: GalaxyViewModel by viewModels()
    private val args: GalaxyDetailFragmentArgs by navArgs()

    override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
        viewLifecycleOwner.lifecycleScope.launch {
            val galaxy = viewModel.getGalaxyById(args.galaxyId)
            if (galaxy != null) {
                binding.tvDetailName.text = galaxy.name
                binding.tvDetailDesc.text = galaxy.description

Glide.with(requireContext()).load(galaxy.imageUrl).into(binding.imgDetail)
                activity?.title = galaxy.name
            }
        }
    }
}
```

Step 9: Login & SharedPreferences

Keep UserPrefs class for login as Week 9 lecture note.

- **SplashActivity checks:**

```
if (userPrefs.isLoggedIn && userPrefs.stayConnected) {
    startActivity(Intent(this, MainActivity::class.java))
} else {
    startActivity(Intent(this, LoginActivity::class.java))
}
```

Room is not used for login, only galaxy data.