Lecture 4: Advanced Kotlin & Best practices (Coroutines, Flows, MVVM, Dependency Injection)

Exercise Submission on 22/10/2025

Objectif

Understand and extend an existing Android project built with **Kotlin**, **Jetpack Compose**, **MVVM**, and **Hilt** by analyzing the provided source code and implementing new CRUD features

Expected Outcome

By the end of this assignment, you should:

- Fully understand the **MVVM data flow** (Entity → DAO → Repository → ViewModel → UI)
- Be able to build and connect multiple **Compose screens**
- Practice **Hilt DI** and **Room** integration
- Reinforce good coding and documentation practices

Deadline

- **Submission:** In 1 week (22/10/2025)
- **Deliverables:**
  - Commented source code (zip)
  - Short documentation (1–2 pages) explaining:
    - The architecture used
    - The data flow
    - Difficulties encountered

Subject

A small mobile application implementing a **partial SCRUD** (Search, Create, Read, Update, Delete) has been developed.

The app is based on the following MLD (Data Model):

Student(idStudent, lastName, firstName, dateOfBirth, gender)
Course(idCourse, nameCourse, ectsCourse, levelCourse)
Subscribe(#idStudent, #idCourse, score)

The part related to the **Student** entity is **mostly completed** and serves as a reference for you.

You will:

1. Analyze and comment the existing code.
2. Extend it to include the Course and Subscribe entities.
3. Use debugging to understand the app's data flow and interactions.

The source code is available on moodle : Click here

Only the module « scrudstudents » matters

Part 1 – Understanding the "Student" Module

Comprehend how the Student CRUD is implemented following the MVVM architecture and Jetpack Compose principles.

1. Open all files related to the **Student** entity:
   - StudentEntity.kt
   - StudentDao.kt
   - StudentRepository.kt / StudentRepositoryImpl.kt
   - StudentViewModel.kt
   - StudentListScreen.kt / StudentFormScreen.kt
   - Any related **Navigation** or **Module (Hilt)** files.
2. Add detailed **comments** in each file to explain:
   - The **purpose** of each class and function.
   - The **role** of the variables and parameters.
   - The **data flow** (from database to UI).
   - The **Compose states** and how recomposition works.
3. You may use **breakpoints and the debugger** to see:
   - When and how data changes.
   - When composables recompose.
   - When coroutines are triggered.

Explain especially how StateFlow and collectAsState() work in the ViewModel/UI connection.

Part 2 – Implement CRUD for "Course"

Reuse and adapt the Student module structure to create a **Course** module.

1. Create new files (copy Student ones and rename):
   - CourseEntity.kt
   - CourseDao.kt
   - CourseRepository.kt
   - CourseViewModel.kt
   - CourseListScreen.kt

- CourseFormScreen.kt
2. Use the following types:
   - idCourse: Int
   - nameCourse: String
   - ectsCourse: Float
   - levelCourse: String (values: P1, P2, P3, B1, B2, B3, A1, A2, A3, MS, PhD)
3. Add the CRUD to the navigation graph.
4. Ensure the interface follows **Material Design 3**.

challenge: Add validation (e.g., ectsCourse > 0).

Part 3 – CRUD for "Subscribe"

Link Student and Course entities through a Subscribe table, allowing score management.

1. Create:
   - SubscribeEntity.kt
   - SubscribeDao.kt
   - SubscribeRepository.kt
   - SubscribeViewModel.kt
   - SubscribeListScreen.kt / SubscribeFormScreen.kt
2. Use :
   - idStudent and idCourse as foreign keys.
   - score: Float
3. The form must allow:
   - Selecting a **student** and a **course** from dropdown menus.
   - Entering or updating the **score**.
4. Add the screen to navigation.

challeng : Display the student's name and course title instead of IDs in the list view.