Lab Assignment #3 – Developing Robust Android Applications with MVVM, Navigation, Co-routines and Data Persistence with ROOM

Due Date: Mid-night (11.59 pm) **12th Nov 2025** **Marks/Weightage: 30/15%**

End Date: Mid-night (11.59 pm) ***15th Nov. with 20% late penalty. No Exceptions***

Note: You are required to demonstrate the assignment as per the scheduled lab session as announced by your teacher. 25% penalty for not demonstrating the assignment

IDE: Android Studio – Meerkat Feature Drop Version and Kotlin Jetpack Compose

Purpose: The purpose of this lab assignment is to:

- Implement network calls using Kotlin Coroutines
- Implement navigation between different Compose destinations.
- Pass arguments between destinations within an Android app.
- Design responsive navigation for foldables and large screens.
- Utilize the resizable emulator to test navigation on various screen sizes.
- Set up and configure the Room library in an Android project.
- Create entities, DAOs, and the database for storing data locally.
- Implement methods for saving, reading, updating, and deleting data using Room.
- Implement the MVVM architectural pattern for better code organization and maintenance.
- Utilize the repository pattern to abstract data access and management.

References: Textbook, ppt slides, class examples, and Android tutorials (**https://developer.android.com/develop/ui/views/theming/look-and-feel**). This material provides the necessary information that you need to complete the exercises.

Be sure to read the following general instructions carefully:
- This assignment must be completed in pairs by all the students. This is based on pair programming.
- You will have to **demonstrate your solution in a scheduled lab session** and upload the solution on **luminate** through the **assignment link under Assessments**.

Android Project Naming Rules:

Step 01: You must name your Android Studio **project** according to the following rule:
yourfullname_COMP304*SectionNumber*_Labnumber
For Example: **johnsmith_COMP304Sec001_Lab03. Save location drive name can be C:\COMP304\Assignments or D:\COMP304\Assignments etc.**
If you have more than one exercise in the assignment, then you need to create separate projects for each exercise.

Step 02: Submission rules

Once you complete, run and test projects for all the exercises, then submit your projects as one **zip file and it should be** named according to the following rule:

yourfullname_COMP304SectionNumber_Labnumber.zip.
Example: **johnsmith_COMP304Sec001_Lab03.zip (if your section is 001)**

Exercise 1: Improve/Enhance the ProductApp covered in the class as per following instructions: ***[5 marks]***

Fig 1.0

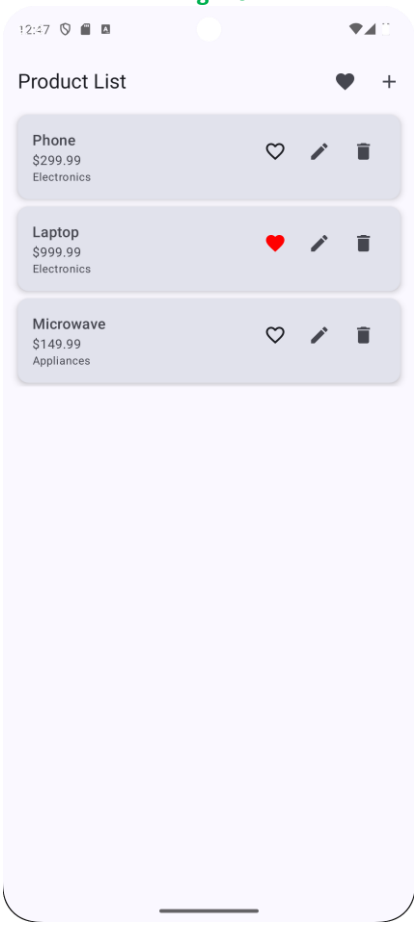
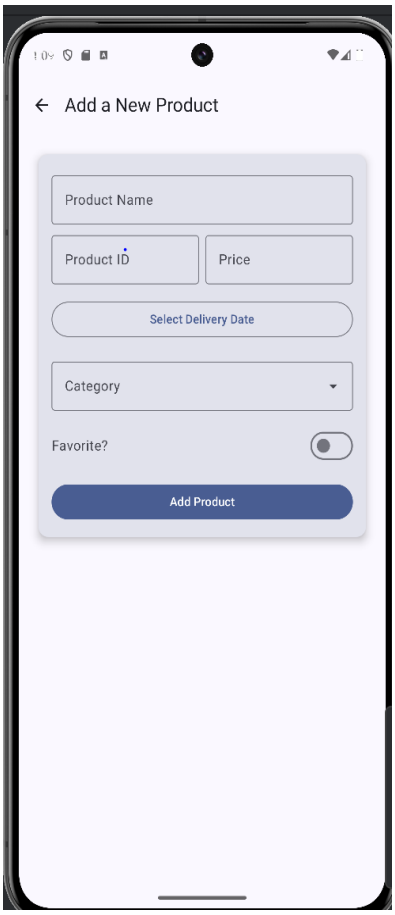


Fig 2.0



1. In Fig 1.0 screen above, add a Search text box to search for a product based on Product ID
2. In Fig 2.0 screen, add another text box for Quantity of product. Quantity > 0

Exercise 2:

You will develop a new app to help users manage their collection of movies DVD efficiently with a visually appealing interface and advanced features. The app should include the following functionalities:

NOTE: You are free to add any extra activity/class which you think necessary.

a) A home screen displaying a list of stored movies. Store at least three movies initially in Room DB (SQLite) ***[5 marks]***
b) A screen to create and add to a persistent store (database) display (using Lazy Column) a new movie. (Movie should have following properties: ***[10 marks]***

1. ID- 2 digits long (11-99)
2. Title
3. Name of Director
4. Price of DVD, should be positive
5. Date of Release, you must define date composable
6. Duration in minutes such as 115 minutes.
7. Genre (Family, Comedy, Thriller, Action etc.) should be a drop down
8. Favorites (On/Off button to make a given movie as your favorite movie)

Note: Unless the above values are provided, a movie should not be added to the database. Also, you can add a duplicate movie to the database. It should compare movies based on ID before adding a new movie to database. ***[5 marks]***

c) A screen to view, edit and delete existing movies. ***[5 marks]***

d) **Display** button to display all the movies stored in the database ***[2.5 marks]***

e) **Favorite** button which only displays the favorites movies ***[2.5 marks]***

f) Responsive layouts and accessibility features.

g) Implementation of MVVM architecture.

h) Use of Jetpack libraries.

Features and Implementations:

1. **Networking with Retrofit(Optional)**:
   - Implement network calls using Kotlin coroutines to handle asynchronous tasks.
   - Handle responses and errors effectively in network operations.
2. **Jetpack Navigation**:
   - Implement navigation between different Compose destinations.
   - Pass arguments between destinations within the app (e.g., passing location details).
   - Design responsive navigation for foldables and large screens.
   - Utilize the resizable emulator to test navigation on various screen sizes.
3. **Local Data Storage with Room**:
   - Set up and configure the Room library in the project.
   - Create entities, DAOs, and the database for storing favorite locations and weather data locally.
   - Implement methods for saving, reading, updating, and deleting data using Room.
4. **App Architecture (MVVM and Repository Pattern)**:
   - Implement the MVVM architectural pattern.
   - Create ViewModel classes for managing UI-related data and business logic.
   - Utilize the repository pattern to abstract data access and management.

Evaluation table:

Item	Percentage of Total Mark	Details
Functionality:	**80%**	
Correct implementation of Retrofit and Kotlin coroutines:		
Co-routine setup and asynchronous handling	20%	Ensure Retrofit is set up correctly and network calls are handled using Kotlin coroutines.

Item	Percentage of Total Mark	Details
Correct implementation of Jetpack Navigation:		
Navigation between Compose destinations	10%	Implement navigation and argument passing between different screens.
Implementation of responsive UI and Room library:		
Responsive navigation and local data storage	30%	Design responsive navigation layouts and configure Room for local data storage.
Correct implementation of MVVM and Repository Pattern:		
ViewModel and repository setup	20%	Ensure ViewModel classes manage UI data and repositories abstract data access.
Friendliness:	**15%**	
Alignments of UI controls	10%	UI controls should be properly aligned and organized, providing a visually appealing layout.
Friendly I/O	5%	The app should provide a user-friendly interface with intuitive input/output operations.
Comments, Correct Naming of Variables, Methods, Classes, etc.	**5%**	Code should be well-documented with appropriate comments. Variables, methods, and classes should follow proper naming conventions.
Total	**100%**	