MA D - Lab 4

Objective 1: Understanding SharedPreferences

Theory: SharedPreferences in Android allow the storage and retrieval of key-value pairs of primitive data types. It's typically used for saving user preferences or small sets of data.

Step-by-Step Guide in Kotlin with Jetpack Compose:

1. **Initialization**: Obtain a SharedPreferences instance. In a Jetpack Compose application, you might do this in your `MainActivity` or a ViewModel.

```
val sharedPreferences = this.getSharedPreferences("MyPrefs", Context.MODE_PRIVATE)
```

2. **Writing Data**: Use an Editor to write data to SharedPreferences.

```
val editor = sharedPreferences.edit()
editor.putString("key", "value")
editor.apply()
```

3. **Reading Data**: Retrieve data from SharedPreferences.

```
val value = sharedPreferences.getString("key", "default value") ?: "default value"
```

4. **Composable Function to Display SharedPreferences Data**: Create a Composable function that displays data stored in SharedPreferences.

```
@Composable
fun ShowPreferencesData() {
    val context = LocalContext.current
    val sharedPreferences = context.getSharedPreferences("MyPrefs",
Context.MODE_PRIVATE)
    val data = sharedPreferences.getString("key", "default value") ?: "default
value"
    Text(text = "Stored value is: $data")
}
```

Expected Output: A simple Jetpack Compose screen with an input field to enter data, a button to save the data to SharedPreferences, and a text element to display the saved data.

Objective 2: Understanding Preferences Datastore

Theory: Preferences Datastore is a modern data storage solution that uses Kotlin coroutines and Flow for storing and retrieving key-value pairs asynchronously, providing a safer and more consistent approach compared to SharedPreferences.

Step-by-Step Guide in Kotlin with Jetpack Compose:

1. **Setup Preferences Datastore**: Define a Preferences Datastore instance.

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

MA D - Lab 4

2. **Writing and Reading with DataStore**: Use DataStore to asynchronously save and retrieve data.

```
// To save a value
val EXAMPLE_COUNTER = preferencesKey<Int>("example_counter")
suspend fun incrementCounter() {
    context.dataStore.edit { settings ->
        val currentCounterValue = settings[EXAMPLE_COUNTER] ?: 0
        settings[EXAMPLE_COUNTER] = currentCounterValue + 1
    }
}

// To read a value
val exampleCounterFlow: Flow<Int> = context.dataStore.data
    .map { preferences ->
        preferences[EXAMPLE_COUNTER] ?: 0
    }
```

3. **Displaying DataStore Values in Compose**: Create a Composable function to display the value from DataStore.

```
@Composable
fun ShowDataStoreValue(exampleCounterFlow: Flow<Int>) {
    val counterValue by exampleCounterFlow.collectAsState(initial = 0)
    Text(text = "Counter value is: $counterValue")
}
```

Objective 3: Understanding Storages (Internal and External)

Theory: Android offers options for internal and external storage to save sensitive data, media files, and more. Internal storage is private to your application, while external storage is accessible by other apps and the user.

Step-by-Step Guide:

- **Internal Storage**: Writing and reading files in internal storage.

```
// Writing to a file
context.openFileOutput("example.txt", Context.MODE_PRIVATE).use {
    it.write("Example content".toByteArray())
}

// Reading from a file
val content = context.openFileInput("example.txt").bufferedReader().useLines
{ lines ->
    lines.fold("") { some, text ->
        "$some\n$text"
    }
}
```

- **External Storage**: Checking for external storage availability and writing/reading files.

MA D - Lab 4

```
// Ensure external storage is available
if (Environment.getExternalStorageState() == Environment.MEDIA_MOUNTED) {
    val file = File(context.getExternalFilesDir(null), "example.txt")
    file.writeText("Example content")
}
```

MA D - Lab 4

Objective 4: Practice Tasks

1. **SharedPreferences Task**: Create a Jetpack Compose screen where users can save and retrieve their favorite color using SharedPreferences.
2. **DataStore Task**: Implement a counter application using Preferences Datastore. The app should increment and display the counter value each time the user presses a button.
3. **Internal Storage Task**: Write an application that saves user notes to internal storage. Provide functionality to save new notes and list all saved notes.
4. **External Storage Task**: Develop a simple media gallery using external storage. The app should list images stored in a specific directory on external storage.