**Assignment 2 *Android Sensor Framework***

COMP2430-Mobile Computing Technologies

Department of Computer Science

Winter 2025

**Group 12**

**Group Members:-**

1. Sanskriti Dabhade (1225131)
2. Sweety Das (1238456)
3. Kirtan Sukhadiya (1222983)
4. Julia Petterle (1185004)

## **Introduction:-** This report presents our group’s submission for Assignment 2 in COMP2430. We extended our Assignment 1 "Greeting App" (named BusinessCardApp) to incorporate the Android Sensor Framework. The updated app now displays the current light level, proximity level, and magnetometer availability in a Column layout using Jetpack Compose, replacing the static business card content from Assignment 1. This document details our implementation, includes a screenshot of the code changes, and provides links to the required material.

**Implementation:-**

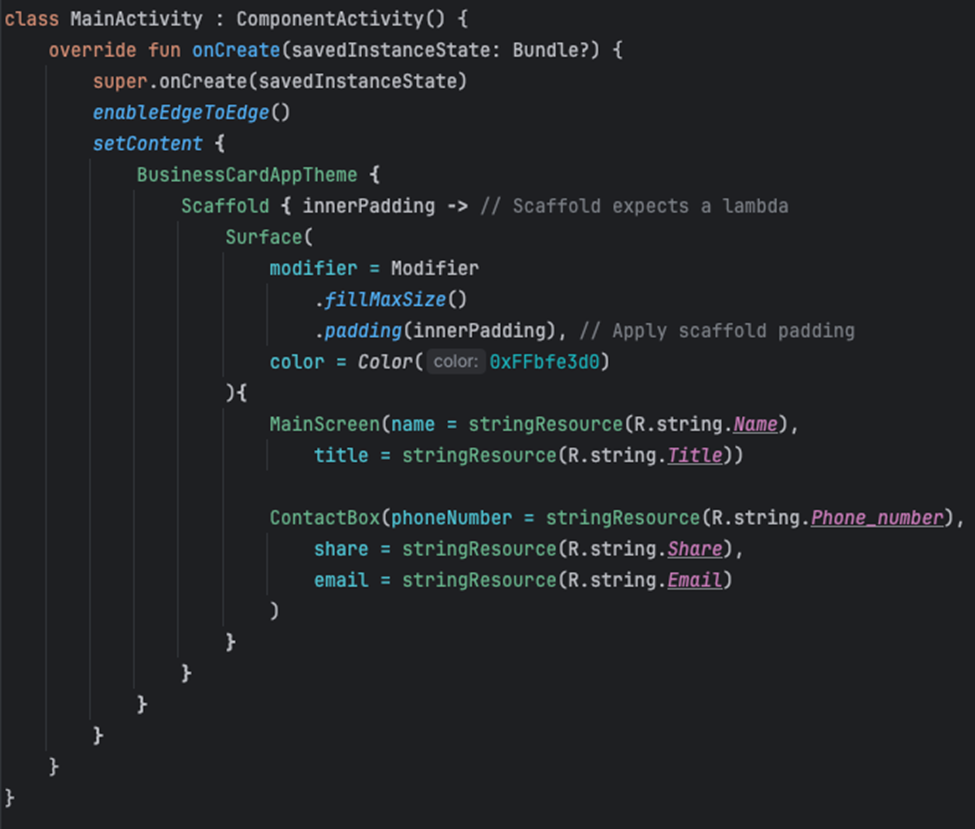
In Assignment 1, our BusinessCardApp displayed static business card information, including a name, title, phone number, share text, and email, using string resources and Jetpack Compose. The UI was rendered via the MainScreen composable, called within the setContent block of MainActivity. For Assignment 2, we modified this app to display:

1. **Current Light Level**: Measured in lux from the light sensor.
2. **Current Proximity Level**: Measured in centimeters from the proximity sensor.
3. **Magnetometer Availability**: A message indicating whether a magnetometer is present on the device.

### **Code Modifications:-**

We extended our Assignment 1 code by replacing the static business card content with sensor data functionality. The MainActivity was also updated to implement SensorEventListener for handling sensor events. Below are the key changes:

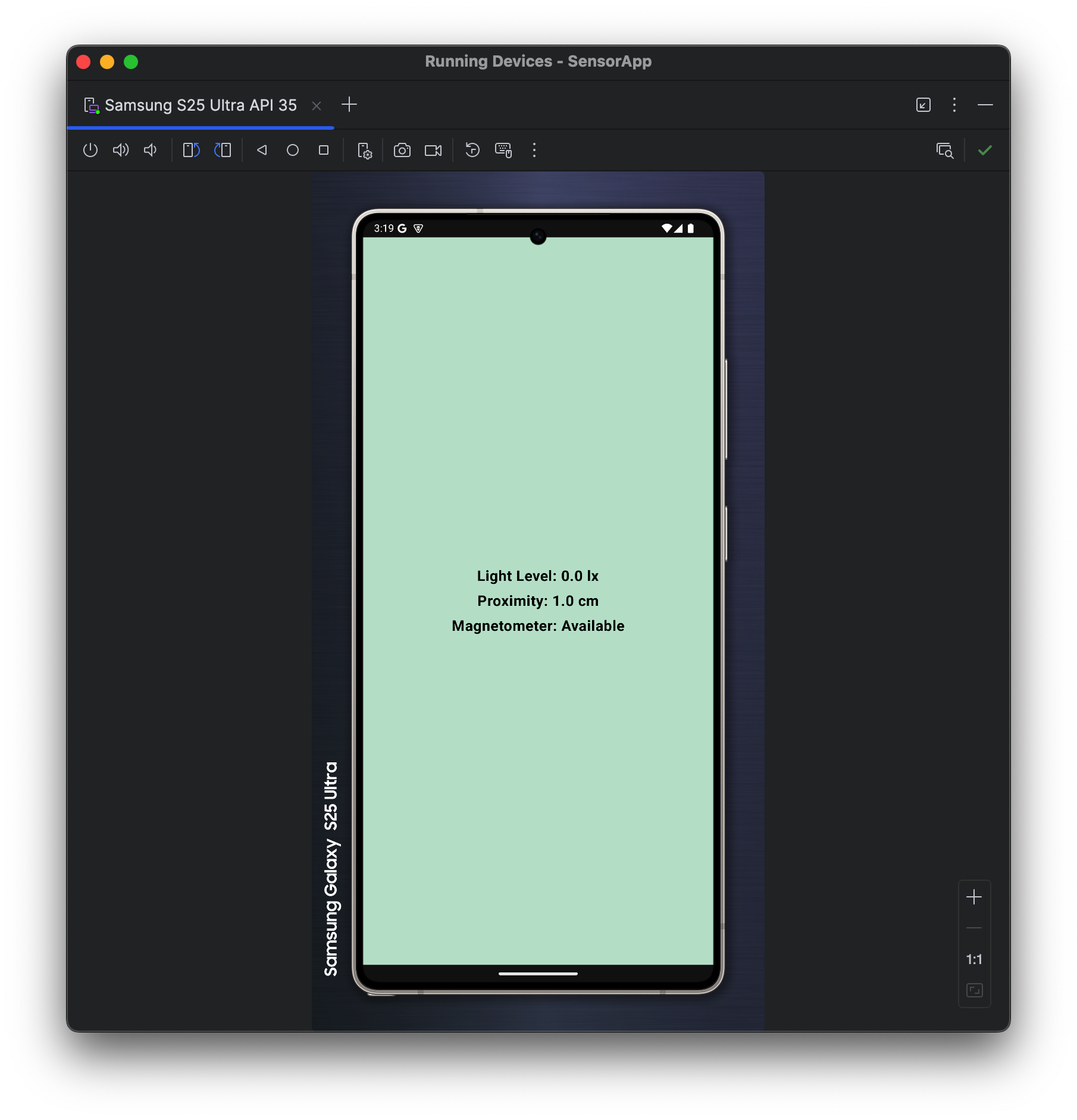
1. **Replacing Static Content with Sensor Data**:  
    In Assignment 1, the setContent block of MainActivity called the MainScreen composable to display static business card details:



For Assignment 2, we replaced the MainScreen composable with SensorInfoScreen, which displays the sensor data. We also added state variables to hold the sensor values and passed them to SensorInfoScreen. The updated setContent block from MainActivity.kt showing the transition:

1. **Sensor Initialization and Management**:  
    We modified MainActivity to implement SensorEventListener and added sensor handling logic. The SensorManager was initialized in onCreate to set up sensors for light, proximity, and magnetometer. The onSensorChanged method updates the state variables (\_lightLevel, \_proximityLevel) with real-time sensor data, while the magnetometer’s availability is determined during initialization.
2. **UI Implementation**:  
    The new SensorInfoScreen composable arranges the sensor data in a Column with centered alignment, bold text, 20.sp font size, and black color. Spacers were added for better spacing between text elements.

*Note: Explanation of the entire code is provided within the Presentation video further.*

**Working Application (Screenshot):-**

## **Running the Application:-**

### **Setup Instructions**

To run our app:

1. Open the attached project files in Android Studio.
2. Connect an Android device or set up an emulator (API 21 or higher).
3. Build and run the app using Android Studio’s "Run" option.
4. Grant any necessary permissions for sensor access if prompted.

### **Demonstration**

We recorded a 15-second screen recording of the app running on an emulator. The video demonstrates:

* The light level updating (simulated using emulator sensor controls).
* The proximity level updating (simulated similarly).
* A message indicating whether a magnetometer is available.

***Screen Recording Link:*** [***Sensor\_App\_Demo.mp4***](https://drive.google.com/file/d/1NlfLkf0KNFCTsU5x-I3yHn2lbd2Wuraq/view?usp=sharing)

### **Presentation Video**

We prepared a short video with audio narration explaining our code modifications and setup process.

***Presentation Video Link:*** [***Sensor App.mp4***](https://drive.google.com/file/d/1GI2dNhqvtpdHEwEBQv4emldeuOeq93kS/view?usp=drive_link)

## **Challenges and Resolutions**

* **Sensor Availability on Emulators**: Some emulators lacked sensor support, so we tested on a physical device to ensure accurate results.
* **UI Transition**: Replacing the business card layout with sensor data required careful state management. We used *mutable* state variables to ensure the UI updates dynamically with sensor changes.

## **Conclusion**

This assignment enhanced our understanding of the Android Sensor Framework and Jetpack Compose. By extending our Assignment 1 app, we successfully implemented a sensor-based UI in a Column layout