## Milestone 2.7: Local UI Enhancements and Status Monitoring

In this milestone, we enhance the Android app’s local user interface to provide important controls and status indicators directly on the device. Even though the PC orchestrates recording sessions, having on-device feedback is crucial for setup and troubleshooting. The phone’s screen will now serve as a simple dashboard to inform the user of connection states, sensor status, and allow limited manual control if needed. Key improvements include:

* **Status Display:** The main Activity will be expanded to show real-time status info for connectivity and sensors. For example, it will display whether the phone is connected to the PC controller (e.g. showing “**PC:** Connected” or “**PC:** Waiting for PC…”), the current recording state (idle vs. recording), and sensor connectivity (such as **Shimmer** and **Thermal Camera** status). These can be shown with text labels and/or colored icons (green for connected, red for disconnected) for quick glanceability. We will also show the device’s **battery level** on screen, since long recording sessions require monitoring power. The app can periodically query the battery percentage via Android’s battery status API (using the sticky ACTION\_BATTERY\_CHANGED intent) and update a battery indicator[[1]](https://developer.android.com/training/monitoring-device-state/battery-monitoring#:~:text=int%20level%20%3D%20batteryStatus.getIntExtra%28BatteryManager.EXTRA_LEVEL%2C%20,1). This way, the user or operator can immediately see if the device needs charging during use.
* **Manual Recording Controls:** We will add local **Start/Stop Recording** buttons in the UI as a fallback control. These controls will invoke the same internal services/methods that the PC commands use to begin or end a recording session. In normal operation (when connected to the PC), these buttons will be hidden or disabled to prevent conflicts, since the PC is the primary controller. However, if the PC connection is not active (e.g. during standalone testing or if the PC software crashes), an operator can still start or stop the recording from the phone. This provides a safety net for development and ensures the system can be used in a basic capacity without the PC. The app logic will ensure that when the PC is controlling the session, local manual controls are overridden or grayed-out to avoid any race conditions.
* **On-Device Camera Preview (Thermal/RGB):** If feasible, the app will show a small live preview of the camera feed on the phone’s screen to assist with framing and focus. This could be a view showing either the RGB camera image, the thermal image, or a combined overlay (depending on what the **Topdon thermal camera** provides). We might include a toggle or tab to switch the preview between RGB and thermal views. Implementing this preview could leverage Android’s camera view utilities – for instance, using a PreviewView from CameraX library, which automatically streams the camera output to a UI widget[[2]](https://developer.android.com/media/camera/camerax/preview#:~:text=When%20adding%20a%20preview%20to,and%20rotated%20for%20proper%20display). If the camera output is high resolution (e.g. 4K for recording), we will use a downscaled stream for the on-screen preview to maintain performance. One approach is to repurpose the frames used for the PC streaming/recording: as those frames are captured, create a reduced-size bitmap and display it in an ImageView on the Activity. This ensures minimal overhead, since we’re using existing frame data. The preview on the device is mainly to help a local operator align the camera correctly (useful if someone is physically holding or positioning the device), whereas the primary operator interface is on the PC. Even so, having this visual feedback on the phone itself is valuable for quick adjustments.
* **Calibration Capture Feedback:** To assist in calibration procedures (e.g. capturing images of a calibration target at different angles), the phone’s UI will provide immediate feedback whenever a calibration photo is taken. For example, when the PC triggers a calibration capture, the phone can briefly flash the screen or display a message like “Calibration photo captured!” so the person handling the device knows the capture was successful. A simple implementation is to show a **Toast** message – a small popup that auto-dismisses – confirming the action[[3]](https://developer.android.com/guide/topics/ui/notifiers/toasts#:~:text=A%20toast%20provides%20simple%20feedback,automatically%20disappear%20after%20a%20timeout). We might also play a subtle camera shutter sound to give an audible cue (Android’s MediaActionSound API provides a built-in shutter click sound effect[[4]](https://stackoverflow.com/questions/13069345/android-play-camera-shutter-sound-programmatically#:~:text=27)). These cues will signal the user to hold still momentarily and then move to the next position if multiple calibration shots are needed. If the calibration process involves step-by-step guidance, the PC software will likely handle instructions, but the device at minimum will acknowledge each capture. This feedback loop improves user confidence that the app received the command and performed the capture, which is especially important in multi-angle calibration routines.
* **Settings and Configuration Screen:** We will add a simple settings interface to make the app’s configurable parameters easily editable without modifying code. Accessible via a menu option or a separate Settings activity, this screen will allow the user (or developer) to set values such as the **PC server IP address/port**, the **Shimmer device MAC address** for Bluetooth, default recording parameters (e.g. video resolution, frame rate), and other relevant options. Android provides built-in support for such preference screens[[5]](https://www.geeksforgeeks.org/android/how-to-implement-preferences-settings-screen-in-android/#:~:text=In%20many%20apps%2C%20we%20have,preferences%20setting%20screen%20in%20Android), typically using a PreferenceFragment or similar to list settings and save them via SharedPreferences. When the app starts, it will load these preferences – for example, to automatically connect to the configured PC IP or to initiate the Shimmer connection using the saved MAC. Having a settings UI means we don’t need to recompile or hard-code for small changes, and end-users can adjust certain behavior (like pointing the app to a different server or swapping out sensor devices) in a user-friendly way. All settings will be persisted and validated (e.g. ensuring IP format is correct, etc.) to avoid misconfiguration. This addition makes the system more flexible and user-centric, as non-developers can tweak key parameters safely.

Overall, these UI enhancements will make the Android app much more **transparent and user-friendly** in operation. The on-device display of connection status, sensor status, and battery gives immediate insight into what the app is doing and whether everything is functioning (crucial when multiple devices are running in parallel). The local controls and feedback mechanisms serve as both a development aid and a fail-safe for users, ensuring that even if the orchestrating PC has issues, the device can still operate and inform the user. In summary, Milestone 2.7 focuses on refining the user interface so that both developers and end-users can confidently monitor and control the multi-sensor recording process directly on the device when needed, complementing the PC-based control system. This will lead to a more robust and **easy-to-use platform** for data collection.

**Sources:**

1. Android Developers – Monitoring Battery Level (Retrieving battery percentage from ACTION\_BATTERY\_CHANGED)[[1]](https://developer.android.com/training/monitoring-device-state/battery-monitoring#:~:text=int%20level%20%3D%20batteryStatus.getIntExtra%28BatteryManager.EXTRA_LEVEL%2C%20,1)
2. Android Developers – CameraX PreviewView (adding an in-app camera preview UI)[[2]](https://developer.android.com/media/camera/camerax/preview#:~:text=When%20adding%20a%20preview%20to,and%20rotated%20for%20proper%20display)
3. Android Developers – Toasts (brief on-screen messages for operation feedback)[[3]](https://developer.android.com/guide/topics/ui/notifiers/toasts#:~:text=A%20toast%20provides%20simple%20feedback,automatically%20disappear%20after%20a%20timeout)
4. Stack Overflow – Using MediaActionSound for camera shutter audio feedback[[4]](https://stackoverflow.com/questions/13069345/android-play-camera-shutter-sound-programmatically#:~:text=27)
5. GeeksforGeeks – Implementing an Android Settings/Preferences Screen[[5]](https://www.geeksforgeeks.org/android/how-to-implement-preferences-settings-screen-in-android/#:~:text=In%20many%20apps%2C%20we%20have,preferences%20setting%20screen%20in%20Android)

[[1]](https://developer.android.com/training/monitoring-device-state/battery-monitoring#:~:text=int%20level%20%3D%20batteryStatus.getIntExtra%28BatteryManager.EXTRA_LEVEL%2C%20,1) Monitor the Battery Level and Charging State  |  App quality  |  Android Developers

<https://developer.android.com/training/monitoring-device-state/battery-monitoring>

[[2]](https://developer.android.com/media/camera/camerax/preview#:~:text=When%20adding%20a%20preview%20to,and%20rotated%20for%20proper%20display) Implement a preview  |  Android media  |  Android Developers

<https://developer.android.com/media/camera/camerax/preview>

[[3]](https://developer.android.com/guide/topics/ui/notifiers/toasts#:~:text=A%20toast%20provides%20simple%20feedback,automatically%20disappear%20after%20a%20timeout) Toasts overview  |  Android Developers

<https://developer.android.com/guide/topics/ui/notifiers/toasts>

[[4]](https://stackoverflow.com/questions/13069345/android-play-camera-shutter-sound-programmatically#:~:text=27) audio - Android: Play camera shutter sound programmatically - Stack Overflow

<https://stackoverflow.com/questions/13069345/android-play-camera-shutter-sound-programmatically>

[[5]](https://www.geeksforgeeks.org/android/how-to-implement-preferences-settings-screen-in-android/#:~:text=In%20many%20apps%2C%20we%20have,preferences%20setting%20screen%20in%20Android) How to Implement Preferences Settings Screen in Android? - GeeksforGeeks

<https://www.geeksforgeeks.org/android/how-to-implement-preferences-settings-screen-in-android/>