Chapter 1

Chapter 3: Requirements and System Analysis

1.1 3.1 Problem Statement and Research Context

Modern physiological monitoring techniques often rely on **Galvanic Skin Response (GSR)** sensors attached directly to a subject's skin to measure electrodermal activity. While GSR is a proven indicator of stress and arousal, traditional contact-based measurement is intrusive and limits natural behavior. The research problem addressed is how to **predict GSR in a contactless manner** using alternative sensing modalities (such as thermal imaging and visual cameras) without sacrificing accuracy. In the current state of physiological measurement, cameras and thermal sensors have advanced to capture subtle physiological cues (e.g. facial temperature changes or perspiration) that could correlate with stress. However, **no integrated system existed** to simultaneously collect *synchronized* thermal, visual, and reference GSR data required to develop and validate contactless GSR prediction models. This thesis operates in the context of **affective computing and human-computer interaction research**, where unobtrusive monitoring of stress and emotional state is highly desirable. The goal is to provide a multi-sensor recording platform that enables new experiments in which participants are monitored **without wires or attached electrodes**, facilitating more natural interactions (e.g. in social or virtual reality settings) while still capturing high-quality ground-truth physiological data.

To bridge this gap, the project developed a **Multi-Sensor Recording System for Contactless GSR Prediction**. The system is designed to collect **synchronized multi-modal data streams** — specifically high-resolution visual video, thermal infrared imagery, and **GSR readings from a Shimmer sensor** — in real time. By aligning these data streams with sub-millisecond precision, the system creates rich datasets for training and evaluating machine learning models that estimate stress (or related physiological signals) from camera data alone. This research context demands a solution that is *both* scientifically rigorous (accurate timing, reliable signals) and practical for field use (mobile devices, untethered subjects). In summary, the problem statement centers on building a **distributed data acquisition system** that can capture synchronized physiological and imaging data to enable **contactless GSR measurement** research. The remainder of this chapter details the requirements derived from this problem and the system analysis that shaped the solution.

1.2 3.2 Requirements Engineering Approach

The requirements for the multi-sensor system were derived using a combination of stakeholder-driven analysis and iterative prototyping. **Stakeholder analysis** identified the primary stakeholders as: (1) **Research scientists** who require accurate, high-fidelity data and straightforward operation in experiments; (2) **Study participants** who benefit from unobtrusive monitoring (hence the need for contactless methods and minimal encumbrances); (3) **Technical maintainers/developers** of the system who need the software to be maintainable, extensible, and testable; and (4) **Institutional review boards / ethics committees**, concerned with data security and participant safety. Each stakeholder group introduced distinct requirements. For example, researchers emphasized data synchronization accuracy and multi-modal integration, participants motivated requirements for comfort and privacy, and developers focused on modular architecture and high code quality standards to ensure reliability.

Given the experimental nature of the project, the **requirements engineering approach was iterative and incremental**. The team followed an agile-like process: initial core requirements were established from the research objectives (e.g. *"record thermal and video data synchronized with GSR"*), and a **prototype system** was quickly built to validate feasibility. As the prototype was tested with actual sensors, new requirements and refinements were discovered (for instance, the need for automated device re-connection on failure, or a method to log stimulus events during recording). The repository's commit history reflects these iterations — each commit often corresponded to implementing or refining a specific requirement (e.g. adding the Shimmer sensor integration or improving time synchronization). This evolutionary process ensured continuous alignment between requirements and implementation.

The project also adhered to established **software requirements specification practices** (in line with IEEE guidelines) to systematically document each requirement with an identifier, description, and priority. Requirements were classified into functional and non-functional categories for clarity. Throughout development, a strong emphasis was placed on **validation and testing** to ensure requirements were met: a comprehensive test suite (with 95was developed to automatically verify that each functional requirement (device communication, data recording, etc.) behaved as expected in real scenarios. In summary, the requirements engineering combined up-front analysis of research needs with continuous feedback from implementation, resulting in a robust set of requirements that guided the system design and implementation.

1.3 3.3 Functional Requirements Overview

Table 3.1 lists the **Functional Requirements (FR)** identified for the multi-sensor recording system. Each requirement is labeled with a unique ID and a priority (H = High, M = Medium) indicating its importance. These functional requirements capture the intended capabilities and behaviors of the system. They were derived to ensure the system meets the needs of coordinating multiple devices, acquiring various sensor data streams, synchronizing and storing data, and supporting the research workflow.

Table 3.1 — Functional Requirements

ID Functional Requirement Description Priority FR-01 **Centralized Multi-Device Coordination:** The system shall provide a PC-based master controller application that can connect to and manage multiple remote recording devices (Android smartphones). This enables one operator to initiate and control recording sessions across all devices from a single interface. H

FR-02 **User Interface for Session Control:** The PC master controller shall offer an intuitive graphical user interface (GUI) for configuring sessions, displaying device status, and controlling recordings (start/stop). The GUI should show connected device indicators and allow the user to easily monitor the recording process in real time. H

FR-03 **High-Precision Synchronization:** The system shall synchronize all data streams (video frames, thermal frames, GSR samples) with a unified timeline. Recording on all devices must start nearly simultaneously, achieving time alignment with an accuracy on the order of 1 millisecond or better. Each data sample/frame will be timestamped to enable precise cross-modal H correlation [?].

FR-04 **Visual Video Capture:** Each Android recording device shall capture high-resolution **RGB video** of the participant during the session. The system should support at least 30 frames per second at HD (720p) resolution or higher (up to the device's capabilities, e.g. 1080p or 4K) for detailed visual data. The video recording is to be continuous for the session H duration, saved in a standard format (e.g. MP4).

FR-05 **Thermal Imaging Capture:** If a recording device is equipped with a thermal camera, the system shall capture **thermal infrared video** in parallel with the RGB video. Thermal frames must be recorded at the highest available resolution and frame rate (device-dependent) and time-synchronized with other streams. This provides contactless skin temperature data H corresponding to the participant's physiological state.

FR-06 **GSR Sensor Integration:** The system shall integrate **Shimmer GSR sensor** devices to collect ground-truth physiological signals (electrodermal activity and related sensors). The PC controller must handle Shimmer data either via direct Bluetooth connection or via an Android device acting as a relay (proxy) for the H sensor [?]. All connected GSR sensors should be managed concurrently, and their data samples (GSR conductivity, plus other channels like PPG or accelerometer) timestamped and synchronized with the session timeline.

FR-07 **Session Management and Metadata:** The system shall allow the user to create a new *recording session* and automatically assign it a unique Session ID. During a session, the controller will maintain metadata including session start time, configured duration (if applicable), and the list of active devices/sensors. Upon session start, each device and sensor is H registered in the session metadata, and upon stop, the session is finalized with end time and duration recorded [?] [?]. A session metadata file (e.g. JSON or text) shall be saved, summarizing the session details for future reference.

FR-08 **Local Data Storage (Offline-First):** All recording devices shall store their captured data **locally on-device** during the session to avoid reliance on continuous network streaming. Video streams are saved as files on the smartphones (and any PC-local video source) and GSR data is logged (e.g. to CSV) on the PC or device collecting it. Each data file is H timestamped or contains timestamps internally. This *offline-first* design ensures no data loss in case of network disruption and maximizes reliability of recording.

FR-09 **Data Aggregation and Transfer:** After a recording session is stopped, the system shall support automatic aggregation of the distributed data. The PC controller will instruct each Android device to **transfer the recorded files** (video and any other data) to the PC over the network. The files are transmitted in chunks with verification — the PC confirms the file M sizes and integrity on receipt [?]. All files from the session are collected into the PC's session folder for centralized storage. (In the event automatic transfer fails or is unavailable, the system permits manual retrieval as a fallback.)

FR-10 **Real-Time Status Monitoring:** The PC interface shall display real-time status updates from each connected device, including indicators such as recording state (recording/idle), battery level, storage space, and connectivity health [?]. M During an active session, the operator can observe that all devices are recording and see any warnings (e.g. low battery) in real time. Optionally, the system may also show a low-frame-rate preview of the video streams for verification purposes (e.g. a thumbnail update) [?]. These status and preview updates help the operator ensure data quality throughout the session.

FR-11 **Event Annotation:** The system shall allow the researcher to **annotate events** or markers during a recording session. For example, if a stimulus is presented to the participant at a specific time, the researcher can log an event (through the PC app UI or a hardware trigger). The event is recorded with a timestamp (relative to session start) and a short description M or type [?]. All such events are saved (e.g. in a `stimulus_events.csv` in the session folder) to facilitate aligning external events with physiological responses during data analysis.

FR-12 **Sensor Calibration Mode:** The system shall provide a mode or tools for calibration and configuration of sensors before a session. This includes the ability to capture calibration data for the cameras (e.g. a one-time procedure to spatially align the thermal and RGB cameras using a reference pattern) and to configure sensor settings (focus, exposure, thermal range, M etc.) as needed. Calibration data (such as images of a checkerboard or known thermal target) are stored in a dedicated calibration folder for each session or device [?] [?]. This requirement ensures that the multi-modal data can be properly registered and any sensor biases corrected in post-processing.

FR-13 **Post-Session Data Processing:** The system shall support optional **post-processing steps** on the recorded data to enrich the dataset. For example, after a session, a *hand segmentation* algorithm can be run on the recorded video frames to identify and crop the participant's hand region (since GSR is often measured on the hand). If enabled, the PC controller will M automatically invoke the hand segmentation module on the session's video files and save the results (segmented images or masks) in the session folder [?]. This automates part of the data analysis preparation (e.g. extracting relevant features) and is configurable by the user.

Discussion: The above functional requirements cover the core capabilities of the system. Together, they ensure that the **multi-sensor recording system can capture synchronized data from multiple devices and sensors** and manage that data effectively for research use. The design addresses multi-device coordination (FR-01, FR-02) and tight time synchronization (FR-03) as top priorities, since these are critical for aligning different data modalities. Requirements FR-04 through FR-06 enumerate the data acquisition needs for each sensor modality — visual video, thermal imaging, and GSR — reflecting the system's multi-modal nature. Session handling and data management (FR-07, FR-08, FR-09) form the backbone that guarantees recordings are organized and preserved reliably (e.g., creating session metadata and using offline local storage to avoid data loss). Real-time feedback and control (FR-10 and FR-11) improve the usability of the system during experiments, allowing the operator to monitor progress and mark important moments. Finally, FR-12 and FR-13 address advanced functionality: calibration support and post-processing, which enhance the quality and utility of the collected data (these are considered "Medium" priority since the system can run without them, but they are valuable for achieving research-grade results). Many of these requirements are explicitly supported by the implementation — for instance, the ShimmerManager class in the code confirms the multi-sensor integration and error-handling for GSR sensors [?] [?], and the session management logic creates metadata files and directory structures as specified [?] [?]. The next section will consider constraints and quality attributes (non-functional requirements) that also had to be satisfied to meet these functional goals.

1.4 3.4 Non-Functional Requirements

In addition to the explicit features and behaviors, the system must fulfill several **Non-Functional Requirements (NFR)** that define qualities such as performance, reliability, and usability. Table 3.2 summarizes the key non-functional requirements for the multi-sensor recording system, again with unique IDs and priority levels. These requirements ensure the system not only *works*, but works effectively and robustly in the contexts it will be used (research labs, possibly mobile or field environments, with human participants involved).

Table 3.2 — Non-Functional Requirements

ID Non-Functional Requirement Description Priority NFR-01 **Real-Time Performance:** The system shall operate in real time, handling data streams without undue delay. All components must be efficient enough to **capture video at full frame rate and sensor data at full sampling rate** without buffering issues or frame drops. For example, the Android High app should sustain 30 FPS video recording and $\tilde{5}0$ Hz GSR sampling simultaneously. The end-to-end latency from capturing a sensor sample/frame to logging it with a timestamp should be minimal (well below 100 ms), ensuring a responsive system.

NFR-02 **Synchronization Accuracy:** The system's clock synchronization and triggering mechanisms shall be precise, as reflected in FR-03. The design should ensure that any timestamp discrepancies between devices are below acceptable thresholds (on the order of milliseconds). In practice, this may High involve time synchronization protocols or timestamp calibration. Each data sample is tagged with both device-local time and a unified time reference to permit alignment during analysis [?]. This requirement guarantees the **temporal integrity** of the multi-modal dataset.

NFR-03 **Reliability and Fault Tolerance:** The system must be reliable during long recording sessions. It shall handle errors gracefully — for instance, if a device temporarily disconnects (due to network drop or power issues), the system will attempt to reconnect automatically and continue the High session without crashing [?]. Data already recorded should remain safe in such events (thanks to local storage on devices). The system should not lose data or corrupt files even if an interruption occurs; any partial data is cleanly saved up to the point of failure. Robust **error handling and recovery** mechanisms are implemented (e.g., retry logic for device connections and file transfers).

NFR-04 **Data Integrity and Accuracy:** The system shall ensure the integrity and accuracy of recorded data. All data files (videos, sensor CSV) are verified for correctness after recording — for example, the file transfer process includes confirming file sizes and sending High acknowledgments [?]. Time stamps must be consistent and accurate (no clock drift over typical session durations). The GSR sensor data should be sampled with stable timing and recorded with the correct units (e.g., microSiemens for conductance) without clipping or quantization errors. This requirement is crucial for the scientific validity of the collected dataset.

NFR-05 **Scalability (Multi-Device Support):** The architecture should scale to **multiple recording devices** operating concurrently. Adding more Android devices (or additional Shimmer sensors) to a session should have a linear or manageable impact on performance. The network and PC controller must Medium handle the bandwidth of multiple video streams and sensor feeds. At minimum, the system is expected to support a scenario of *at least two Android devices* plus one or two Shimmer sensors recording together. The design (threaded server, asynchronous I/O, etc.) allows scaling up the number of devices with minimal modification [?].

NFR-06 **Usability and Accessibility:** The system's user interface and workflow shall be designed for **ease of use** by researchers who may not be software experts. This means the PC application should be straightforward to install and run, and the process to start a session is simple (e.g., High devices auto-discover the PC, one-click to start recording). Visual feedback (FR-10) is provided to reduce user uncertainty. The Android app should require minimal user interaction — ideally launching and automatically connecting to the PC. Clear notifications or dialogs guide the user if any issues occur (e.g. permission requests, errors). The system should also be documented well enough that new users can learn to operate it quickly.

NFR-07 **Maintainability and Extensibility:** The software shall be designed following clean code and modular architecture principles to facilitate maintenance and future extension. For example, the Android app follows an MVVM (Model-View-ViewModel) architecture with dependency injection (Hilt) to Medium separate concerns, making it easier to modify or upgrade components (such as replacing the camera subsystem or adding a new sensor) without affecting others. The PC code similarly separates the GUI, networking, and data management logic into distinct modules. Code quality metrics were enforced (e.g., complexity limits) to keep the implementation understandable and testable [?] [?]. Additionally, a high level of automated test coverage was achieved, so developers can confidently refactor the system and add features while catching regressions early. This requirement ensures the longevity of the system as a research platform.

NFR-08 **Portability:** The system should be portable and not dependent on specialized or expensive hardware beyond the sensors themselves. The PC controller is a cross-platform Python application that can run on standard laptops or desktops (the only requirement being moderate processing power, $\tilde{4}$ Medium GB RAM, and Python 3.8+ environment). The Android app runs on common Android devices (Android 8.0 or above) and supports a range of phone models, provided they have the needed sensors (camera, etc.) and Bluetooth for Shimmer. This allows the system to be deployed in different laboratories or even off-site (using a router or local hotspot for networking). Portability also implies that the system's components (PC and mobile) communicate over standard interfaces (TCP/IP network, JSON messages) without requiring wired connections, increasing the flexibility of where and how it can be used.

NFR-09 **Security and Data Privacy:** While the system is typically used in controlled research settings, basic security measures are required. The network communication (JSON command channel and file transfers) should occur only over a secure local network. Only authorized devices (which send a Medium correct handshake/hello message with expected format) are allowed to connect to the PC controller, reducing the risk of unauthorized interception. Additionally, participant data (video and physiological signals) are sensitive, so the system should provide options to encrypt stored data or otherwise protect data at rest, according to institutional data handling guidelines. (For example, the recorded files can be stored on encrypted drives or be anonymized by not embedding personal identifiers.) Although full encryption of live streams is not implemented in the current version, this requirement is noted for completeness to ensure ethical research data management.

Discussion: These non-functional requirements underline the system's quality attributes that make it suitable for research use. Performance (NFR-01) and synchronization accuracy (NFR-02) ensure that the **data quality** meets scientific standards — the system can capture high-resolution, high-frequency data in sync, which is essential for meaningful analysis. Reliability and data integrity (NFR-03, NFR-04) are critical given that experimental sessions are often unrepeatable — the system must not crash or lose data during a trial. Scalability (NFR-05) acknowledges that the research may expand to more devices or participants; the system's **distributed architecture** (with a multi-threaded server and modular device handling) was designed to accommodate this growth with minimal rework. Usability (NFR-06) was a high priority because complex setups can impede researchers — features like auto device discovery and real-time feedback were included to make the system as user-friendly as possible. Maintainability and extensibility (NFR-07) have been addressed by following rigorous software engineering practices (the commit history shows extensive refactoring and documentation efforts to keep code complexity in check [**?**]). This means the system can be updated (for example, to integrate a new type of physiological sensor or to improve algorithms) by future developers with relative ease. Portability (NFR-08) ensures the system can be used in various environments — whether in a lab or a field study — without heavy infrastructure. Finally, security (NFR-09) and data privacy considerations reflect the ethical dimension: while not the primary focus during development, the design allows operation on closed networks and the addition of security layers so that sensitive participant data is safeguarded. In sum, the NFRs complement the functional requirements by guaranteeing that the system operates **efficiently, robustly, and responsibly** in a real-world research context.

1.5 3.5 Use Case Scenarios

To illustrate how the system is intended to be used, this section describes several primary **use case scenarios**. These scenarios represent typical workflows for the multi-sensor recording system, demonstrating how the functional requirements come together to support research activities. *(Figure 3.1 provides a use case diagram summarizing the actors and interactions in these scenarios*

Placeholder

.) The main actor in these use cases is the **Researcher** operating the system via the PC Master Controller, with secondary actors being the **Recording Devices** (Android phones with cameras, sensors) and the **Participant(s)** being recorded. The scenarios assume all devices have been set up and the software is running on the PC and smartphones.

Figure 3.1: Use case diagram illustrating the system's primary scenarios — conducting a multi-participant recording session, performing system calibration, and real-time monitoring with event annotation (Placeholder).

Use Case 1: Multi-Participant Recording Session — *"Conduct a synchronized multi-sensor recording for one or more participants."* In this primary use case, a researcher records an experimental session involving physiological monitoring. The steps are as follows:

1. **Setup:** The researcher ensures all Android devices (e.g., two smartphones, each focusing on a participant) are powered on and running the recording app. Each participant is equipped with a Shimmer GSR sensor (e.g., worn on the fingers) which is either paired to the PC or to one of the phones. All devices are connected to the same Wi-Fi network. The researcher launches the PC Controller application and sees indications that the devices have auto-discovered and connected (each device sends a "hello" registration to the PC [?], and appears in the PC UI list of available devices). The PC UI shows each device's ID and capabilities (for example, "Device A: camera + thermal, Device B: camera + GSR").

2. **Initiate Recording:** The researcher configures session parameters on the PC (optionally setting a session name or notes) and clicks "Start Session". The PC controller then broadcasts a **start recording command** to all connected devices [?]. Each Android device receives the command and begins recording its sensors: the cameras start capturing video frames and writing to local MP4 files, and if a device has a paired Shimmer, it starts streaming GSR data to a local file. The PC concurrently might start its own recording (e.g., if a webcam on the PC is used as another video source). All these actions are synchronized — devices either start immediately upon command or according to a coordinated start timestamp so that their internal clocks align (the system accounts for network latency by using very short command messages and preparing devices in advance). The PC UI updates to indicate "Recording in progress" for each device, and the session timer begins.

3. **Data Collection:** During the session (which could last e.g. 5 minutes, 30 minutes, or longer), the system continuously collects data. Each smartphone writes video frames to its storage and periodically sends status updates to the PC (battery level, elapsed recording time, file size, etc. as per FR-10). The Shimmer sensors stream GSR (and possibly additional signals like PPG, accelerometer) — if a Shimmer is connected directly to the PC, the PC logs

that data in real time; if connected via a phone, the phone relays the sensor data packets to the PC over the network or stores them to include in its file. The system ensures that all data streams are timestamped consistently (each device uses a monotonic clock or synchronized timestamp). If any device encounters an error (for example, a camera error or a Bluetooth disconnection), it automatically tries to resolve it (restart the camera, reconnect the sensor) without user intervention, as long as the session is active. The researcher observes the PC dashboard occasionally — for example, they might see a small preview frame updating for each device, confirming that video is being captured, and status text like "Device A: Recording 120s, Battery 85

4. **Concurrent Participants:** This use case can involve **multiple participants**. For instance, if two participants are in an interaction, each is being recorded by a separate phone and wearing a separate GSR sensor. The PC coordinates both streams. This scaled scenario simply repeats the above for each device — because of the system's scalability, it handles two sets of data as easily as one. The PC's session metadata will log both devices under the same Session ID, and all data will share the timeline. The researcher can thus capture social or group scenarios with synchronized physiological measurements for each person.

5. **Stop Session:** Once the desired recording duration or experimental task is completed, the researcher clicks "Stop Session" on the PC. The controller sends a **stop command** to all devices. Each Android device stops recording: it finalizes the video file (closing the file safely) and similarly finalizes any sensor data file. The devices then each report back a final status (e.g., "Saved 300s of video, file size 500 MB"). At this point, the PC invokes the data aggregation process (FR-09): it requests each device to transmit its files. One by one, the phones send their recorded files to the PC: for example, Device A sends `session_20250806_101530_rgb.mp4` and `thermal.mp4`, Device B sends its `session_20250806_101530_rgb.mp4` and `sensors.csv`. The PC receives these via the JSON socket connection in chunks [?] [?], writing them into the PC's own storage (`recordings/session_20250806_101530/DeviceA_rgb.mp4`, etc.). The PC verifies the file sizes match what the device reported. When all files are received, the PC marks the session as completed, updates the session metadata (end time and duration) [?], and presents a summary to the researcher (e.g., "Session completed: 2 devices, 2 video files, 1 sensor file, duration 5:00"). The researcher can then proceed to analyze the data offline.

6. **Post-conditions:** The outcome of this use case is that **all relevant multi-modal data has been recorded and centralized**. The session folder on the PC now contains subfolders or files for each device: e.g., video files from each camera, thermal data, GSR CSV, plus a session log and metadata. The researcher has a complete, time-synchronized dataset from the multi-participant session, which can be used for model training or other analysis.

Use Case 2: System Calibration and Configuration — *"Calibrate sensors and configure system settings prior to recording."* This secondary use case is often performed before an actual data collection session (it can be considered a preparatory or maintenance scenario). Its goal is to ensure that all sensors are correctly configured and aligned to collect high-quality data.

1. **Camera Calibration:** The researcher wants to calibrate the alignment between the RGB camera and the thermal camera on a device (if the device has both, or between two devices if one provides thermal and another RGB). Using a **calibration module** in the system (invoked via

the PC UI or directly on the device app), the researcher places a known reference object (such as a checkerboard pattern or a thermal reference pad) in view. They then trigger a **calibration capture** — the system might capture a set of images from the RGB and thermal cameras simultaneously. These images are saved in the session's calibration folder (e.g., `calibration/` directory for that session) [?] [?]. Later, the researcher can use these image pairs to compute a spatial transformation that maps thermal images to the RGB frame (this computation might be done by an external script or a provided calibration tool). The resulting calibration parameters (e.g., a matrix or alignment file) can be stored and used in analyzing the recorded data (so that features in the thermal and visual data can be compared pixel-to-pixel after alignment).

2. **Sensor Configuration:** The system allows adjusting certain sensor settings to suit the experimental needs. For example, the researcher can configure the Shimmer GSR device's **sampling rate** (e.g. 51.2 Hz by default, but maybe set to 128 Hz for higher resolution) and which channels are enabled (GSR, photoplethysmograph, 3-axis accelerometer, etc.) [?] [?]. This is done through a configuration interface (the PC or Android app exposes options if the sensor is connected). The chosen configuration is saved so that the device will use those settings in the upcoming session — the ShimmerManager class, for instance, stores a profile for the device with its MAC address and the enabled channels and sampling rate [?]. Similarly, the researcher can set the resolution or frame rate for the smartphone cameras if needed (though by default, the system auto-selects the highest supported resolution and a standard frame rate).

3. **System Checks:** Before recording, the researcher performs a quick system diagnostic. They may use a **"Preview" mode** in the PC UI where each device streams a preview frame or short segment to the PC. The PC displays these to confirm the cameras have the participant in frame and the thermal camera is properly focused, etc. The researcher also checks that all devices show good battery levels or are connected to power (to satisfy reliability needs for a long session). The PC's status panel might show available storage on each device; if a device has low space, the researcher knows to free up space (this is part of configuration — ensuring enough memory for the session).

4. **Result:** After this use case, the system is calibrated and configured. All devices are aligned and set with optimal parameters. This increases the quality of the data in the main recording use case. Calibration images and configuration files are stored for reference. For instance, the `session_config.json` might record the settings used (thermal camera frame rate, emissivity setting, etc.) [?] [?], and the `calibration/` directory holds raw frames used for calibration. The system is now ready to begin an actual recording session with confidence that data from different sensors will be comparable and that sensors are operating within desired ranges.

Use Case 3: Real-Time Monitoring and Annotation — *"Monitor live data and annotate events during a session."* This use case occurs concurrently with an active recording (as in Use Case 1) but focuses on the researcher's interaction with the system while it is running. Its purpose is to allow the researcher to mark important moments and observe the data quality in real time.

1. **Live Monitoring:** As the session proceeds, the researcher watches the PC application's live dashboard. The PC receives periodic **preview frames** from the devices (for example, a downsampled image every few seconds) [?]. The UI might show a small video window for each

device updating with these frames, so the researcher can ensure the participant remains properly framed and that lighting/thermal conditions are good. The PC also could display running plots for sensor streams (e.g., a real-time chart of GSR values) — in this system, since GSR is recorded either on PC or relayed, the PC can plot the incoming GSR signal live. This real-time feedback helps detect any issues (e.g., a sensor detached or a camera view obstructed) so they can be fixed immediately rather than only discovered after the session.

2. **Stimulus/Event Annotation:** During the recording, certain events might occur that the researcher wants to log. For instance, in a stress experiment, at time 2:00 a loud sound might be played to startle the participant. The researcher clicks an **"Add Event"** button in the PC UI at that moment (and perhaps types "Startle sound" or selects an event type from a list). The PC then records an event with a timestamp (relative to session start) and a label "Startle sound". In the implementation, this triggers a call to the Session Manager's `addStimulusEvent()` method on the Android device or PC, which appends the event to the `stimulus_events.csv` file along with the exact timestamp [**?**]. Multiple events can be logged: e.g., "Questionnaire start", "Questionnaire end", "Unexpected noise", etc., each at specific times. These annotations are invaluable later when analyzing the physiological data, as they mark when external stimuli or notable participant actions occurred.

3. **Adaptive Control (if applicable):** In some scenarios, the researcher might make adjustments during the session. For example, if they notice the thermal camera's auto-calibration has triggered a re-adjustment (which might briefly pause the feed), they could note that or disable auto-calibration next time. Or if one participant leaves the frame, the researcher might physically adjust the camera and use the preview to recentre. The system design allows for such mid-session interventions without stopping the recording — the data continues to be captured uninterrupted.

4. **Observation of Limits:** Real-time monitoring also lets the researcher see if any **system limits** are being approached — for instance, the PC might show that one phone's storage is 90or that its battery is at 15stop the session a bit early or ensure data just up to that point is used. Because of NFRs like reliability, these warnings are part of the UI to prevent data loss (e.g., a low storage warning at runtime might prompt the researcher to stop recording before the file system is full).

5. **Session End and Event Log:** After the session, the `session_metadata.json` or `session_log.txt` on each device/PC includes a summary of any events annotated (with their timestamps and labels) [**?**] [**?**]. The researcher, when reviewing the data, can easily line up the event times with spikes in GSR or changes in thermal imagery. The result of this use case is an **enhanced dataset**: not only the raw sensor data, but also contextual markers that help interpret that data. The live monitoring aspect ensured that the data captured is of high quality (since the operator could catch problems in real time), and the annotation aspect enriched the dataset for analysis.

These use case scenarios demonstrate the end-to-end flow of how the system is used in practice. In a typical experiment day, the researcher would first calibrate and configure the system (Use Case 2), then run one or multiple recording sessions (Use Case 1) while monitoring and annotating (Use Case 3), and finally end up with well-organized data ready for analysis. The scenarios involve multiple system components interacting seamlessly: for example, the **network communication** plays a crucial role in all cases (device discovery, start/stop commands, live data relay, file transfer) and must function reliably under the hood. The next section will delve into the system's architecture and data flow, explaining how these use cases are supported by the design of the software and the distribution of responsibilities between the PC controller and the Android devices.

1.6 3.6 System Analysis (Architecture & Data Flow)

This section analyzes the overall architecture of the multi-sensor recording system and describes the flow of data through the system's components. The design follows a **distributed client-server architecture** with the PC as the central server (or coordinator) and the Android devices as clients. It also employs modular subsystems to handle the various concerns: user interface, device communication, sensor data handling, and data storage. The analysis here shows how the chosen architecture meets the requirements (both functional and non-functional) and how data moves from capture to storage in a synchronized way. Key architectural elements and their interactions are summarized in *Figure 3.2* and the data flow is illustrated in *Figure 3.3*.

Figure 3.2: High-level system architecture, showing the PC Master Controller communicating over a Wi-Fi network with multiple Android Recording Devices and directly or indirectly with Shimmer GSR sensors (Placeholder).

Architecture Overview: The system architecture can be viewed in two layers — **hardware nodes** and **software components**. On the hardware side, we have: (1) the **PC Master Controller** (a laptop/desktop running the Python application) and (2) one or more **Android Recording Devices** (smartphones). Optionally, (3) **Shimmer GSR sensor devices** are also present; they can interface either with the PC (via Bluetooth) or with an Android phone (via the phone's Bluetooth). The PC and phones are connected via a **Wireless LAN** (e.g., a dedicated Wi-Fi router or hotspot), forming a private network for the system. This network enables low-latency communication required for synchronization and data transfer.

On the software side, the PC runs a **Master Controller Application** which includes several components working together: a **GUI Module** (for the user interface), a **Session Manager** (for session metadata and file management), and a **Network Communication Server** (to handle connections with devices). The network server on PC is implemented as a custom JSON socket server listening on a known port (e.g., 9000) [?] [?]. It accepts incoming connections from the Android clients and uses a length-prefixed JSON message protocol to exchange commands and data. Each connected Android device is represented in the PC software as a **RemoteDevice** object which tracks its capabilities (camera, thermal, GSR, etc.) and status [?] [?]. The PC's Session Manager on the other hand handles higher-level logic: creating session folders, writing metadata, and coordinating the start/stop across devices.

Each **Android Recording Device** runs an **Android app** (written in Kotlin) that has its own internal architecture. The app is built with a **Model-View-ViewModel (MVVM)** pattern. The core components include: a **Recording Controller/Service** (which manages the camera and sensor hardware on that phone), a **Session Manager** (on Android, which parallels the PC's session logic, creating local folders on the phone's storage for each session), and a **Network Client** that maintains a socket connection back to the PC. The Android app's UI (if the user opens it) provides local status, but in typical operation it runs mostly headlessly after startup, responding to PC commands. The Android's Recording Controller uses Android's Camera2 API for video and the Shimmer SDK for sensor data if a Shimmer is connected to the phone. Crucially, the Android app does not make autonomous decisions — it acts on commands from the PC or on a local user action (which is rare in this use case). This separation ensures that **control logic is centralized** on the PC, fulfilling the requirement of FR-01 (centralized multi-device coordination).

The **Shimmer GSR Sensors** integration is architected flexibly. The system supports three modes (as mentioned in FR-06): **Direct PC Connection**, **Android-Mediated Connection**, or **Simulation Mode** (for testing). In direct mode, the PC runs a Shimmer Bluetooth driver (via the PyShimmer library) in the Python app — the ShimmerManager class on PC opens a Bluetooth COM port to the Shimmer and reads data packets in a background thread [?] [?]. In Android-mediated mode, an Android phone pairs with the Shimmer (using the Shimmer Android API) and that phone's app becomes responsible for reading the GSR data; the phone then sends those sensor readings to the PC over the network (using JSON messages of type "sensor_data"). The PC doesn't directly talk to the Shimmer in that case; it receives the data already parsed from the phone. The architecture allows both modes to operate simultaneously if needed (for example, two Shimmers could be connected, one via PC, one via a phone). In all cases, the Shimmer data is funneled into the same session structure: the PC will eventually store it as part of the session data (either logging directly or saving the file sent from the phone).

Several design patterns and strategies were used to satisfy maintainability and extensibility (NFR-07) in the architecture. For instance, the Android app relies heavily on **dependency injection** (using Dagger/Hilt): components like the Camera Recorder, Thermal Recorder, and Shimmer Recorder are provided to the Main ViewModel, so they can be easily replaced or mocked for testing. This makes it possible to extend support to new sensor types by adding new modules without altering the core logic. On the PC side, the networking is abstracted behind a `JsonSocketServer` class with well-defined events (device_connected, status_received, etc.), and the GUI is kept separate in a Qt-based `MainWindow`. This separation enforces a **modular architecture** — for example, one could develop an alternative web-based UI (and indeed, the code contains a **`web_ui`** module as an experimental interface) without needing to rewrite how sessions or networking work. The architecture also emphasizes thread separation for performance: the PC networking runs in a background thread (so that heavy file transfers don't freeze the GUI), and the Android uses background threads or coroutines for camera and file operations (preventing UI jank on the phone). Overall, the chosen architecture ensures that the system can reliably coordinate multiple devices and handle data streams, while also being organized for future modifications.

Data Flow Analysis: The flow of data through the system can be described step-by-step for a typical session, highlighting how information moves and transforms from capture to storage. *Figure 3.3 depicts this flow, from the moment the user initiates a session to the final* *data aggregation*

Placeholder

. Here is the sequence:

1. **Session Initiation (Command Flow):** The user action of starting a session on the PC triggers command messages over the network. The PC's JsonSocketServer sends a `{"type": "command", "command": "start_recording", ...}` (for example) to each connected Android device. These messages are small JSON payloads, prefixed with a length header (to ensure the receiver knows how many bytes to read). When an Android device receives the start command, it immediately responds (if needed) with an acknowledgment (`{"type": "ack", "cmd": "start_recording", "status": "ok"}`) and then begins its recording process. This command flow is one-to-many (one PC to multiple clients) and happens almost simultaneously to all devices (the PC either sends commands in a quick loop or uses a broadcast helper function to send to all).

2. **Sensor Data Capture (Local Flow on Devices):** Once recording, each device is capturing data from sensors:

3. The **camera data** (visual and thermal) flows from the camera hardware through Android's Camera2 API into either a file or a buffer. On Android, the CameraRecorder class sets up a MediaRecorder that encodes video frames directly to an MP4 file on the device's file system. Simultaneously, for thermal, if a separate stream exists, it might use a similar approach or raw frames saved as images (depending on implementation). The key is that frames are timestamped by the system — the MediaRecorder frames are implicitly timed, and if raw frames are grabbed (for thermal or for preview), the code attaches the current timestamp to them (for preview frames, they even convert to Base64 strings to send to PC).

4. The **GSR sensor data** flow varies by mode: in direct PC mode, the PC's ShimmerManager class receives Bluetooth packets, decodes them to numeric values (like GSR microSiemens) and immediately writes them to a CSV file or stores them in memory queues [**?**]. In Android-mediated mode, the Shimmer Android API delivers sensor samples to the phone app, which then either writes to a local CSV on the phone or streams the values as JSON messages to the PC (the code supports a streaming socket for live data). In both cases, each sensor sample is augmented with timing info: e.g., the code records the sample's device timestamp and the system time it arrived on the PC [**?**], ensuring later that alignment can be done.

5. **Real-Time Data Communication (Status/Preview Flow):** Throughout recording, a parallel flow of **status and preview data** occurs from the Android devices back to the PC. Each device periodically sends a `status` message with its current recording status (every few seconds). This includes data like battery `timestamp` (which can be used by PC to gauge device clock vs PC clock). These status packets help update the UI and also carry implicit sync info. Additionally, if preview is enabled, the device captures a frame (say every 1 second), compresses it (e.g., JPEG), Base64-encodes it, and sends a `preview_frame` message containing a small image string [**?**]. The PC receives this and emits a signal that the GUI uses to display the new frame. This data flow is one-way from each device to PC and is designed to be low-bandwidth (e.g., a thumbnail image rather than full-frame, to avoid bogging down the network). Meanwhile, if the Shimmer is streaming live via Android, those readings might also be sent continuously in

`sensor_data` messages; however, in practice the PC may choose not to plot every point to avoid overload — it could sample or aggregate before display.

6. **Command and Control Feedback:** The PC can also send other commands during the session — for example, if the user triggers an event annotation, the PC might send a `notification` message to devices (or directly log it on PC). In many cases, the annotation is handled on PC side (since PC knows the session time and can just write to the events file immediately). If devices needed to do something (like flash a light when an event is marked), a message would be sent. In our design, most *mid-session* control is minimal; the heavy command flows are start and stop.

7. **Session Termination and Data Gathering:** When stop command is issued, the data flow reverses for file transfer. Each device, after closing its files, initiates a **file transfer protocol** to send the recorded files to PC. The flow is:

8. Device sends a `file_info` message indicating it is about to send a file, including file name and size. For example, `"name": "rgb_video.mp4", "size": 50012345`.

9. PC prepares to receive by creating a new file in the session folder (`DeviceID_rgb_video.mp4`) and responds (implicitly via ack or readiness).

10. Device then sends a series of `file_chunk` messages, each containing a segment of the file encoded (typically in Base64). The PC decodes each chunk and writes to the file handle. This continues until the whole file is sent (chunks are often a few KB each). The PC's networking layer tracks how many bytes have been received and can log progress.

11. When the device finishes sending, it sends `file_end` message with the file name. PC then closes the file and compares the received byte count to the expected size. If they match, PC logs success and sends back a confirmation (`file_received` message with status ok). If there's a mismatch, PC logs an error and could request a retry (in our code, it at least reports the error; a full retry mechanism could be initiated if needed).

12. This process repeats for each file (each device might have multiple files: e.g., one for video, one for thermal, one for sensor data). The PC can pipeline requests or handle one device at a time. In the current implementation, it likely sequentially requests each expected file from each device to avoid network congestion (with a short delay between as indicated).

13. Throughout this, the Session Manager on PC is aware of incoming files and uses `add_file_to_session(device_id, file_type, path, size)` to update the session metadata. This means the session's JSON metadata will list, for each device, the files that were collected (with file paths and sizes), confirming that they are now on the PC.

14. **Post-Processing Flow:** If post-processing (FR-13) is enabled, once all raw data is gathered, the PC may invoke additional processing. For instance, the PC might load the recorded video file and run the hand segmentation algorithm. This would generate output files (images or mask videos) which the Session Manager then places into the session folder (e.g., under a `processed/` subdirectory or by appending results next to original files). The data flow here is local on the PC — using OpenCV or other libraries on the saved files. The results are logged (the `post_processing` field in session metadata is updated to true with a timestamp).

15. **Data Storage and Access:** At the end of the data flow, all data resides in an organized manner on the PC. The **session folder** (typically under a `recordings/` directory) contains the

following: video files (named by device and type), sensor data CSV, events log, session metadata JSON, and any calibration or processed data subfolders. For example, one might see:

- recordings/session_20250806_101530/ session_metadata.json DeviceA_rgb_video.mp4 DeviceA_thermal_video.mp4 DeviceB_rgb_video.mp4 DeviceB_sensors.csv stimulus_events.csv calibration/ (folder with calibration images) processed/ (folder with segmented hand images)

  This structure was defined by the requirements and is implemented in code (the Android app creates similar filenames on its side, and the PC adds the device prefix upon receipt). Because each data file is timestamped or accompanied by timestamps internally, a researcher can load, for instance, the `DeviceB_sensors.csv` and the videos and align them using the timestamps. The data integrity checks ensure these files are complete and not corrupted.

16. **Scalability and Data Flow Considerations:** The architecture's data flow is largely **parallel** — each device operates independently for data capture, which is crucial for scalability. The PC coordinates and eventually brings data together. As more devices are added, the network traffic and PC disk I/O grow, but the modular handling (each device in its thread) means the system can scale up to the point where either network bandwidth or PC write speed becomes the bottleneck. Because video files can be large, the file transfer part is the heaviest data flow; the system mitigates issues by writing chunks to disk as they arrive and using a binary encoding to avoid JSON overhead for large data. Also, to maintain performance, intensive tasks like video encoding are done on the devices (leveraging phone hardware encoders), and the PC mainly handles control and file aggregation — this distribution balances load across the system.

In summary, the system's architecture is a **star topology** with intelligent clients, and the data flow is designed to minimize latency and maintain synchronization. The PC orchestrates the process (command flows out, data flows back), which aligns well with the requirement of central control and monitoring. The use of standard formats (MP4, CSV, JSON) in the data flow ensures that once data reaches the PC, it's immediately usable with analysis tools. The combination of real-time communication and local storage means the system is robust: even if the network has a hiccup, each phone still has its data, and it can be transferred later. The thorough session metadata and logging implemented (on both PC and Android) provide traceability — one can trace each step in the data flow from the logs (e.g., see in the PC log that Device A started recording at time X, or that file transfer for file Y completed, etc.). Thus, the architecture and data flow together fulfill the system requirements by enabling **coordinated, synchronous data capture and reliable data unification**.

Figure 3.3: Data flow diagram for a typical session, illustrating command dissemination, local data capture on devices, status/preview feedback, and post-session data collection into the PC's storage

Placeholder

.

1.7 3.7 Data Requirements and Management

The multi-sensor system produces and handles a variety of data types. This section outlines the specific **data requirements**, including data formats, volumes, and how data is managed and stored to ensure integrity and accessibility for analysis.

Data Types and Formats: The primary data generated by the system are: - **Video data:** This includes regular RGB video and thermal video. The video is encoded in a standard compressed format (MPEG-4/H.264 in an MP4 container) on the recording devices to manage file size. Each video file is accompanied by an internal timestamp track (every video frame has a timestamp in the file), and frame rate information is stored. Thermal video, if captured, is also stored as an MP4 (if using a sensor that outputs a stream) or potentially as a sequence of images if the device captures frame-by-frame — in this implementation it was designed to be an MP4 for consistency (FR-05). Typical video resolution for RGB might be 1920×1080 at 30 fps (if the phone supports it, possibly even 4K as configured in code), whereas thermal cameras usually have lower resolution (for example 320×240 or 160×120 at a lower frame rate like 8—30 fps). Regardless, the system handles these as just "video files" — the exact resolution and frame rate used in a session are recorded in the session configuration file [?] for reference. - **Sensor data:** The Shimmer GSR (and associated sensors) produce time-series data. This data is recorded in **CSV (Comma-Separated Values)** format for human readability and easy import into analysis tools. Each row in the CSV represents a sensor sample. For example, a row might contain: *Timestamp, SystemTime, GSR_ Conductance, PPG, Accel_ X, Accel_ Y, Accel_ Z, BatteryLevel.* The `Timestamp` is the device's relative time or sample count, and `SystemTime` could be an absolute UNIX timestamp (to tie it to PC clock). Using CSV ensures that researchers can open the file in Python, MATLAB, Excel, etc. easily. If multiple Shimmer sensors are used, either multiple CSV files are created (one per device) or all data is merged into one file with device identifiers — in our design we create separate files per device to keep things simple (the file naming will include the device ID or name). - **Metadata and logs:** The system generates JSON and text files for metadata. The **session metadata** (JSON) contains structured information about the session: session ID, start/end times, list of devices, files names, and possibly environment info (like app versions). For instance, `session_metadata.json` might have an entry listing each device by ID and the files it produced. Additionally, a human-readable **session_info.txt** is generated on the Android side listing the folder contents and status [?]. This redundancy is intentional for clarity — researchers can quickly read the text summary or use the JSON for programmatic processing. The PC and devices also maintain **log files** (e.g., `session_log.txt`) that record events and any errors with timestamps — these are useful for debugging and audit trails. - **Event annotations:** As discussed, event markers are stored in a simple CSV file (e.g., `stimulus_events.csv`). Each line has an event timestamp (in milliseconds or a readable time format) and a label. This file is managed by the Session Manager (either on PC or device) whenever an event is added [?]. The format is straightforward, ensuring that during analysis one can load this file and overlay events onto signal timelines.

Data Volume and Storage Considerations: The system is expected to handle significant data volumes, especially for video. For example, one minute of 1080p RGB video at 30 fps can be on the order of 60—120 MB when encoded (depending on the scene and compression bit rate). Thermal video tends to be smaller (both in resolution and often lower frame rate, plus uniform scenes compress well), perhaps a few MB per minute. GSR CSV files are relatively tiny in comparison (on the order of kilobytes per minute; e.g., 60 samples per second for 60 seconds is 3600 lines — a few hundred KB at most). Even so, a multi-hour recording could generate multiple gigabytes of data (mostly video), so the system's data requirement is that devices have **sufficient storage** available. The Android app checks available free space on the device storage before and during recording and can warn the PC if space is low. The data management plan recommends using high-capacity SD cards or internal memory on phones, and the PC of course typically has ample disk space.

To manage this, each recording session's data is isolated in its own directory (both on the device during capture and on the PC after aggregation). This not only organizes the data but makes it easier to move or archive entire sessions. If a user needs to free space, they can archive older session folders to an external drive. The naming convention with session timestamps (e.g., `session_YYYYMMDD_HHMMSS`) ensures uniqueness and chronological order. The inclusion of the session name (if the researcher provided one) in the folder or file names helps in identifying the context of the data (for example, `session_20250806_101530_stressTest` if "stressTest" was given as a name).

Data Integrity and Verification: As outlined in NFR-04, the system has built-in measures to verify data integrity. During file transfer, checksums or at least byte counts are compared. After a session, the Session Manager on PC logs a **session summary** that includes whether each expected file is present and its size. For example, the PC log might say "RGB Video: OK (50012345 bytes), Thermal Video: OK (12345678 bytes), Shimmer Data: OK (N bytes)" confirming successful captures. In case a file is missing or incomplete, that is flagged in the log (and the PC UI would alert the user). The system avoids modifying the raw data once recorded — all data files are write-once (append or create-only). If post-processing outputs are generated, they are written as new files rather than altering the originals. This way, the raw recordings remain pristine for analysis or for rerunning analysis with different parameters.

Data Accessibility and Use: The data formats chosen (MP4, CSV, JSON) make the dataset **portable** across analysis environments. Researchers can copy the entire session folder and load it into analysis software. For example, MP4 videos can be played or imported into computer vision pipelines (OpenCV, etc.), and CSV sensor data can be read by pandas or MATLAB. The system's documentation includes a **Technical Glossary** (not reproduced here) which describes each data field and any calibration that has been applied, so analysts understand how to interpret values (e.g., that GSR is in microSiemens, temperature might be in Celsius if thermal camera yields absolute temperature after calibration, etc.). In some cases, calibration results (from Use Case 2) might also produce data (like a camera intrinsic matrix or a mapping file); these are stored in the calibration folder or appended to the metadata JSON so that analysis code can automatically correct the data if needed.

Data Security and Privacy Management: As noted in NFR-09, while the system doesn't inherently encrypt data, it assumes data will be handled on secure storage. If required, the entire `recordings/` directory on the PC can reside on an encrypted drive. Personal identifiers are generally not embedded in file names (device IDs are generic like "phone1" or a device serial, and session IDs are timestamps or user-defined codes, not participant names). This is a conscious decision to keep the data pseudonymized at the file system level. The mapping from session ID to a specific participant or experiment trial would be maintained separately by the researcher (not in the recorded data itself), to protect participant identity if files are shared with others for analysis.

In conclusion, the system's data management strategy creates a **self-contained record** of each session that is easy to navigate and analyze. By structuring the files logically and including metadata and logs, the system meets all requirements for data completeness, integrity, and usability. Even if months later a researcher or a different team examines the files, they should be able to understand the content and trust that it accurately represents what occurred during the session. This careful attention to data requirements and management ensures that the valuable multi-modal data collected by the system can lead to reliable research findings in contactless GSR prediction.