Significant Technical Issues Faced During the Project Development

- **Power Management**: Ensuring all connected components (ESP32, display, vibration motors, keypad, and FSR sensor) received sufficient power without overheating or causing instability.
- **Sensor Sensitivity**: The FSR (Force Sensitive Resistor) provided inconsistent readings due to environmental noise and improper calibration.
- **Vibration Motor Control**: Difficulty in fine-tuning vibration motors to provide the correct intensity without causing excessive noise.
- **Display Integration**: Issues with rendering data on the display due to memory limitations or communication protocol errors (e.g., I2C or SPI misconfigurations).
- **Keypad Debouncing**: Keypad presses were unreliable due to signal noise, leading to false or missed keypress detections.
- **Code Complexity**: Managing multiple components like sensors, display, keypad, and vibration motors made the main codebase complex and hard to debug.
- **Firmware Issues**: Frequent debugging and firmware uploads required due to logical errors in the code.

Troubleshooting and Corrective Measures Taken

- **Power Management**:
    - Added capacitors to smooth out power delivery to components.
    - Used an external power source to handle components with high current demands.
- **Sensor Sensitivity**:
    - Calibrated the FSR sensor by mapping raw data to meaningful force values.
    - Added a software-based noise filter to stabilize readings.
- **Vibration Motor Control**:
    - Implemented PWM (Pulse Width Modulation) to fine-tune vibration intensity.
    - Tested different mounting methods to minimize noise and maximize efficiency.
- **Display Integration**:
    - Optimized the data sent to the display to reduce memory usage.
    - Resolved communication issues by checking wiring and using appropriate pull-up resistors for I2C communication.
- **Keypad Debouncing**:
    - Added software-based debouncing logic to filter out noise.
    - Used interrupts instead of polling for more reliable keypad input detection.
- **Code Complexity**:
    - Modularized the code by creating **separate header files** for each sensor and component.
    - This allowed independent debugging and testing of each component before integrating them into the main project.

- **Firmware Issues**:
    - Used **Serial Monitor** in the Arduino IDE to print variable values, debugging information, and program flow.
    - Implemented conditional `#define DEBUG` flags to enable or disable debug messages dynamically.
    - Enabled **Verbose Output During Compilation** in Arduino IDE preferences to identify code compilation errors and warnings.
    - Employed **board-specific debugging tools** for ESP32, such as ESP Exception Decoder, to analyze crashes and stack traces.

Lessons Learned

- Effective **component power management** is crucial for stable operation in multi-component systems.
- Proper **sensor calibration** and filtering can significantly improve data accuracy.
- Optimizing **communication protocols** is essential to ensure compatibility between hardware components.
- Implementing **software debouncing** and interrupt-driven designs can enhance the reliability of input systems like keypads.
- **Debugging with IDE Features**:
    - The **Serial Monitor** is invaluable for real-time debugging and monitoring of sensor data, program flow, and hardware states.
    - **Verbose Output During Compilation** helps catch syntax errors, memory overflows, and warnings before uploading code.
    - Tools like **ESP Exception Decoder** make it easier to analyze and resolve runtime errors on ESP32 boards.
- Modularized code and debugging tools (e.g., serial monitor) are invaluable for efficient troubleshooting.