Textual Explanation of the Design and Patterns

1. **Overall Architecture**
   Our system separates the **logic** (insulin delivery, CGM simulation, safety checks) from the **UI** (widgets, dialogs, and main window). The **UI** components collect user inputs and display relevant data, while the core classes (PumpController, WarningChecker, UserProfileManager, etc.) handle the main application logic. In this way, each part focuses on its own responsibilities, which makes the code easier to maintain and extend.

2. **Qt Signals/Slots as the Observer Pattern**

   - **Observer (Publisher-Subscriber)**: We rely heavily on Qt's signals/slots mechanism, which implements the Observer pattern. For example, the CgmSimulator class periodically emits a bgUpdated(double) signal each time it generates a new blood glucose reading.
   - **Observers**: Classes like PumpController and CGMGraphWidget "observe" or subscribe to that signal. Whenever bgUpdated is emitted, these observers receive the new BG value instantly, allowing them to respond (log the reading, update the UI graph, or run Control-IQ logic).
   - This design decouples the CGM simulation from the rest of the system. CgmSimulator doesn't need to know who is listening or how they use the BG; it just emits a signal, and any slot connected to it gets the data.

3. **PumpController as a Mediator**

   - **Mediator**: We use a PumpController class to coordinate interactions among various subsystems (e.g. UserProfileManager, HistoryManager, BolusSafetyManager, CgmSimulator). Rather than each of these calling each other in a tangle, PumpController acts as a central "mediator."
   - For instance, when a manual bolus is requested, the UI calls PumpController::requestBolus(...). That method checks safety constraints, updates the history log, references the CGM time, etc., all in one place. This reduces direct interdependencies between those subsystems.