Design Explanation

1. Central Orchestrator (Device)
Role: Initializes, configures, and coordinates all subsystems.
Responsibilities:
- Loads profiles and instantiates singletons (e.g., DataLogger).
- Creates core components: PumpController, UserInterface, CGMReader, BatteryManager, InsulinReserve, Bloodstream, ControlIQAlgorithm, IOBTracker.
- Manages application lifecycle: power on/off, monitoring loop via a QTimer tick (1 tick = 5 minutes simulated).
- Simulation timing**:** 5 minutes of simulated time corresponds to 1 second of real-world time.
- Delegates safety checks and drives UI refreshes each tick.

2. Insulin Delivery Engine (PumpController)
Role: Controls basal and bolus insulin delivery over time.
Key Methods:
- deliverBolus(amount, rate, suppressTime): schedules a bolus; logs via DataLogger.
- pump(Bloodstream*): on each tick, injects basal (currentBasalRate/12) and portions of the active bolus (activeBolusRate/12) into the bloodstream; debits InsulinReserve.
- suspendBolus(), resumeBolus(), triggerEmergencyStop(): enforce safety by halting delivery and emitting signals (bolusCancelled, bolusDeliveryProgress).

Safety: Checks emergencyStopped and bolusSuspended before injecting.

3. Dose Calculation (BolusCalculator)
Role: Computes insulin dose based on glucose, carbs, and user overrides.
Algorithms:
- Correction bolus: max(0, (glucose - target)/correctionFactor)
- Carb bolus: (carbs * carbRatio)/correctionFactor
- Supports split (extended) bolus with user-configured percentages and delays.

UI Integration: Handles input validation, override toggles, and user confirmations via Qt dialogs.

4. Automated Adjustment (ControlIQAlgorithm)
Role: Monitors CGM readings and dynamically updates basal rate.
Behavior:
- Suspends basal if glucose ≤ 3.9 mmol/L.
- Resumes or reverts to profile basal when glucose stabilizes.
- Logs each adjustment through DataLogger.

5. Continuous Glucose Monitoring (CGMReader)
Role: Simulates CGM readings with random variance and insulin absorption.
Mechanics:
- Increases reading by increasePerHour/12 ± volatility each tick.
- Reduces reading based on blood->getIOB() and correctionFactor.
- Reports disconnected (-1) when the error checkbox is checked.

6. Battery Management (BatteryManager)Role: Simulates battery drain and recharging.
Mechanics:
- Drains 0.1% per tick; emits batteryDead() at 0%.

- Critical threshold (0.15) triggers low-battery alerts.

7. Reservoir and On-Board Insulin Tracking
InsulinReserve: Tracks remaining insulin (300 U max, low at ≤ 30 U).
IOBTracker: Maintains time-decay of delivered bolus entries over a duration (default 240 min).

9. Data Logging (DataLogger)
Role: Central log store for events, glucose readings, and insulin doses.
Features:
- Singleton access via instance().
- Persists logs in JSON (logs.json) with load/save/export.
- Emits logsUpdated() to refresh views.

10. Profile Management (Profile + Settings UI)
Profile class: CRUD operations on named profiles (basal rate, carb ratio, correction factor, target glucose), persisted to profiles.json.
Settings screen: Qt widget allowing create, update, delete, select, and save operations with live list refresh.

11. User Interface Flow (UserInterface)
Screens: Stacked widgets for Login, Home, BolusCalculator, Settings, History, and dynamic Alert dialogs.
Navigation: Signals from child screens trigger screen transitions and device unlock.
Synchronization: UI listens to PumpController and BolusCalculator signals to update bolus progress and countdown.

12. Historical View (History)
Role: Displays a searchable, filterable table of past log entries.
Mechanics: Retrieves LogEntry list on logsUpdated() or user queries; filters by text or event type.

<u>State Modeling with UML</u>

The system's dynamic behaviors were modeled with UML state diagrams (referenced in Insulin Pump Simulator - UML State Diagrams.pdf). These diagrams helped us identify key states and transitions such as:
- Normal Operation
- Suspension/Resumption
- Emergency stop

This state-driven approach makes it easier to enforce safety protocols and ensures that the simulator behaves predictably under various conditions.

<u>Safety and ErrorHandling</u>

Fail-safe validation:
- In the PumpController, we added checks (eg., not pumping if in an emergency state or if bolus delivery is suspended) to avoid dangerous insulin over-delivery.

Alerts and Logging:
- The DataLogger captures every event (whether an error, a system alert, or routine activity) to support troubleshooting and historical review. This design supports requirements listed in our traceability matrix (see Traceability-Matrix-draft.pdf) where every safety-critical action is logged.

Use of Qt's Signal and Slot

- Error handling and UI updates are managed through signals and slots, which decouples the presentation layer (UserInterface) from back-end processes (eg., BatteryManager, CGMReader). This helps with safety and debugging.

<u>Traceability and Requirement</u>

The design decisions were not made in isolation but rather mapped directly to our project requirements and use cases. For example:

- UC3: Deliver Manual Bolus
  The BolusCalculator computes the dose, and the PumpController's deliverBolus() and pump() methods handle the gradual delivery over time.
- UC5: Monitor and Adjust Insulin Delivery
  The ControlIQAlgorithm class actively monitors CGM data and triggers adjustments via PumpController.
- UC7: Error handling & Alerts
  Components like BatteryManager and CGMReader include methods (such as alertLowBattery() and alertCGMDisconnected()) that integrate with the UserInterface for proactive error notifications.

<u>Implementation Decisions</u>

- Time Dependent Pumping: The PumpController's pump() method simulates insulin delivery in "ticks". Instead of delivering the entire bolus immediately, it calculates the amount per tick based on a rate.
- Direct Use of Shared Objects: Insulin Reserve, DataLogger and CGMReader are passed as pointers to the PumpController and other classes, ensuring a single source of truth.

Overall, our design emphasizes a modular, traceable, and safe system:

- The Device class remains the central controller, overseeing interactions among all components.
- PumpController is refined to deliver insulin over time in a realistic simulation.
- All design decisions are driven by clearly defined requirements and supported by UML diagrams and traceability matrices.