Basera Farah Abdulle - 101257784
Dulika Gamage - 101263208
Davis Stanko - 101268432
Rae Shi - 101281994

COMP 3004 Team 17 Final Project
Tandem t:slim X2 Insulin Pump Simulator

Team Responsibilities:

- Basera 101257784
    - Make Design Decisions & organize ideas & debug
    - Design and implement updateInsulin() control IQ algorithm for basal insulin behaviour
    - Implement starting, stopping, or Resuming Insulin behaviour
    - Implement insulin graphing behaviour and clean up the final GUI layout
    - Implement history + error logs features
    - Provide user manual documentation notes
    - Contribute to creating use cases & sequence diagrams (N1, N2)
- Davis Stanko 101268432
    - Make Design Decisions & organize ideas & debug
    - Contribute to creating, updating, and finalizing use cases
    - Implement Battery and Insulin Cartridge logic
    - Develop unit tests
    - Create Requirements Traceability Matrix
    - Write textual explanation of design decisions
    - Record video of simulation
- Dulika Gamage 101263208
    - Make Design Decisions & organize ideas & debug
    - Designed/implemented/managed the GUI and helped revise it based on changes
    - Personal profile functionality, disconnect functionality, charge battery functionality, and cleaned up history output logs
    - Contribute to updating/finalizing Use Cases
    - Make Use Case Diagram
    - Make State Machine Diagram
    - Make UML Class Diagram
    - Make Sequence Diagrams (N4, S1, S2, S3)
- Rae Shi 101281994
    - Make Design Decisions & organize ideas & debug
    - Make github repository & draft version of header files and cpp code
    - Implement the bolus calculation functionality
    - Make sequence diagram (N3)

Use Cases:

Use Case 1: Set up and Initialization

Actors: User

Preconditions: User owns t:slim X2 Insulin Pump, device battery is sufficient.

Success guarantee: The pump is ready for insulin delivery.

Trigger: The user decides to set up or restart the pump after a shutdown.

Main Success Scenario:

1. User presses power button
2. Device starts up and displays homescreen
3. Home Screen is displayed
4. User presses power button
5. Device powers off display

Extension:

1.a: If the device does not turn on

- 1.a.1: User must charge the device.

Use Case 2: User Navigating Home Screen

Actors: User

Preconditions: The device is turned on and the home screen is displayed.

Success guarantee: The user obtains real-time updates on the pump's status.

Trigger: User as started up device.

Main Success Scenario:

1. User can monitor the following information
    - Battery life
    - Insulin On Board (IOB - amount & time remaining of any active insulin on board)
    - Status - current pump settings + insulin delivery status
    - Insulin level - amount left in cartridge
    - CGM (Continuous Glucose Monitoring) data graph
2. User can press navigation buttons
    - Bolus calculator button - which directs users to the bolus calculator
    - Personal Profile Settings
    - History Logs
    - Pause/Resume Insulin
    - Options button - which provides access to insulin delivery settings, alerts, and system config settings

Use Case 3: Manage Personal Profiles

Actors: User

Description: The user reads, updates, and applies personal profiles to adjust insulin delivery settings.

Preconditions: The pump is turned on and accessible.

Postconditions:

- The personal profiles are updated according to the user's actions.
- One profile is set as the active profile for insulin delivery.

Success Guarantee: The user can successfully manage personal profiles, and the pump applies the active profile's settings to insulin delivery.

Trigger: The user decides to view existing profiles, modify a profile, or set a different profile as active to adapt insulin delivery to their needs.

Main Success:

1. Navigate to Personal Profiles Section
2. Select Action: The user chooses to perform one of the operations:
    a. Read:
        i. The user views the list of existing profiles (Morning, Afternoon, Night).
        ii. The user can see each profile's detailed settings (e.g., Basal Rate, Correction Factor, Carb Ratio, Blood Glucose).
    b. Update:
        i. User selects an existing profile by clicking its radio button.
        ii. User presses the "Edit" button to modify settings (e.g., changes Basal Rate to 1.5 u/hr).
        iii. User saves changes by pressing the "Submit" button, updating the profile.
    c. Apply Profile
        i. The user selects a profile by clicking its corresponding radio button and clicking submit, designating it as the active profile for insulin delivery.

Extensions:

2.a: Invalid Input

- 2.a.1: The system prevents the user from inputting invalid values (e.g., negative Basal Rate).

Use Case 4: Deliver a Manual Bolus

Actors: User

Description: The user administers a bolus dose via the bolus calculator, which calculates the required insulin dose.

Preconditions: The pump must have available insulin and be functioning properly.

Success guarantee: A bolus dose is delivered as per user input.

Trigger: The user initiates a bolus to cover food intake or correct high glucose.

Main Success:

1. User accesses the Bolus Calculator section on the main interface.
2. User enters their current blood glucose level (e.g., 5.5 mmol/L) and carbohydrate intake (e.g., 45 g)
3. User initiates bolus delivery by pressing the "View Calculation" button.
4. The Bolus Calculator calculates an appropriate dose and the bolus is delivered with 60% immediately and 40% over a set duration (e.g., 3 hours as specified in the Extended Duration fields).
5. The event is logged in the insulin delivery history.

Use Case 5: Start, Stop, or Resume Basal Insulin Delivery

Actors: User, Control IQ Technology

Preconditions: The pump is running, and a personal profile is set (Morning, Afternoon, or Night).

Success guarantee: The basal insulin delivery state is updated according to user input or Control-IQ's automatic adjustments.

Trigger: The user manually starts, stops, or adjusts basal insulin. Control-IQ predicts that insulin adjustment is necessary based on CGM data.

Main Success Scenario:

1. Users basal insulin rate is configured in current personal profile
2. The pump continuously delivers insulin at the specified rate unless interrupted
3. Control-IQ automatically adjusts basal insulin:
    - Decreases insulin delivery if CGM predicts blood glucose ≤ target glucose + 0.03 in 30 minutes
        i. Decrease of 0.1 to 0.5 U/hr from the current basal rate
    - Suspends insulin delivery (0 U/hr) if CGM predicts blood glucose ≤ target glucose-0.1 in 30 minutes.
    - Increases insulin delivery if CGM predicts blood glucose ≥ target glucose + 0.5 in 30 minutes.
        i. Increase of 1.2 to 2.0 U/hr to current basal rate
4. User can pause delivery by pressing the "Pause Insulin" button; the system sets basal rate to 0 and logs the event ("Insulin delivery paused.").
5. User can resume delivery by pressing the "Resume Insulin" button; the system restores the profile's basal rate and logs the event ("Insulin delivery resumed.").
6. Control-IQ records all manual and automatic adjustments into the insulin delivery log/history.

Extensions:

3.a: If Control-IQ predicts hyperglycemia, where current blood glucose ≥ 8.9 mmol/L

- 3.a.1: The pump increases insulin delivery to maintain the maximum safe rate, 2.0 U/hr and logs the warning.

3.b: If Control-IQ predicts hypoglycemia, where current blood glucose < 3.9 mmol/L

- 3.b.1: The pump stops all insulin delivery to maintain a rate of 0 U/hr and logs the warning.

<u>Use Case 6: View Insulin Delivery History</u>

<u>Actors:</u> User

<u>Description:</u> The user reviews a real-time log of system events, including insulin delivery, glucose updates, and device status changes.

<u>Preconditions:</u> The pump simulator application is running.

<u>Postconditions:</u> The user reviews a chronological list of events from the current session.

<u>Success Guarantee:</u> The user sees a live, text-based log of all insulin administration events and system operations.

<u>Main Success:</u>

1. View the Log Panel
2. View Records: The system automatically appends the following event types to the log as they occur:
    a. Bolus Injections: Amount and resulting glucose level (e.g., "Bolus injected: 1.5 | Glucose: 5.2").
    b. Basal Rates: Current rate and glucose trends (e.g., "Basal insulin delivered: 0.12 | Glucose: 5.3 | Predicted: 5.2").
    c. Device Status: Power on/off, battery level changes, and warnings (e.g., "Power on.", "WARNING: Battery level low (10%)").
    d. Glucose Updates: Current glucose values (e.g., "Glucose: 5.3 mmol/L").
    e. Profile Changes: Active insulin profile (e.g., "Profile set to Morning").
    f. Alerts (e.g., "User is hypoglycemic") are embedded in the log as text entries.
3. Close the Log Panel: The user presses the "Close History Logs" button to hide the log panel.

Use Case 7: Handle Pump Malfunctions and Errors

Actors: User, System

Description: The system detects malfunctions or errors (e.g., low battery, low insulin, disconnection, occlusion) and alerts the user through the interface. The user takes corrective action to resolve the issue, or the system suspends insulin delivery for safety.

Preconditions:

- The pump simulator is powered on and previously was operational.
- The pump has encountered an error condition.

Postconditions:

- The issue is resolved, allowing the pump to resume normal operation.
- All error events are logged.

Success Guarantee: The system successfully detects and alerts the user to malfunctions, ensuring safe operation by suspending insulin delivery when necessary, and the user can take appropriate corrective actions.

Trigger: The system detects an error condition, such as a low battery, low insulin level, CGM disconnection, occlusion, or a critical failure like a pump shutdown.

Main Success Scenario:

1. Error Detection: The system identifies a malfunction or error condition, such as:
    a. Low battery (e.g., battery level drops to 10%).
    b. Out of battery (e.g., battery level drops to 0%).
    c. Low insulin (e.g., insulin cartridge has 30 units remaining).
    d. Device disconnection (e.g., user presses "Disconnect Device").
    e. Occlusion (e.g., user presses "Cause Occlusion").
    f. Hypoglycemia/hyperglycemia (e.g., glucose < 3.9 mmol/L or ≥ 8.9 mmol/L).
2. Alert User: The system displays an alert on the pump's home screen with the appropriate symbol (as per the user manual, page 5) and a message describing the issue:
    a. Low battery: "WARNING: Battery level low (10%)."
    b. Out of battery: "System powered off due to battery depletion."
    c. Low insulin: "WARNING: Insulin level low (30 units)."
    d. Device disconnection: "Device disconnected, reconnect device to user."
    e. Occlusion: "Occlusion occurred, check infusion site for blockages."
    f. Hypoglycemia/hyperglycemia: "User is hypoglycemic" or "User is hyperglycemic."
3. Automatic Safety Action: For critical errors that could lead to unsafe insulin delivery (e.g., occlusion, CGM disconnection, or pump shutdown), the system automatically suspends insulin delivery:
    a. The pump stops the simulation timer and sets the state to stopped.
    b. The event is logged (e.g., "Device disconnected, reconnect device to user.").

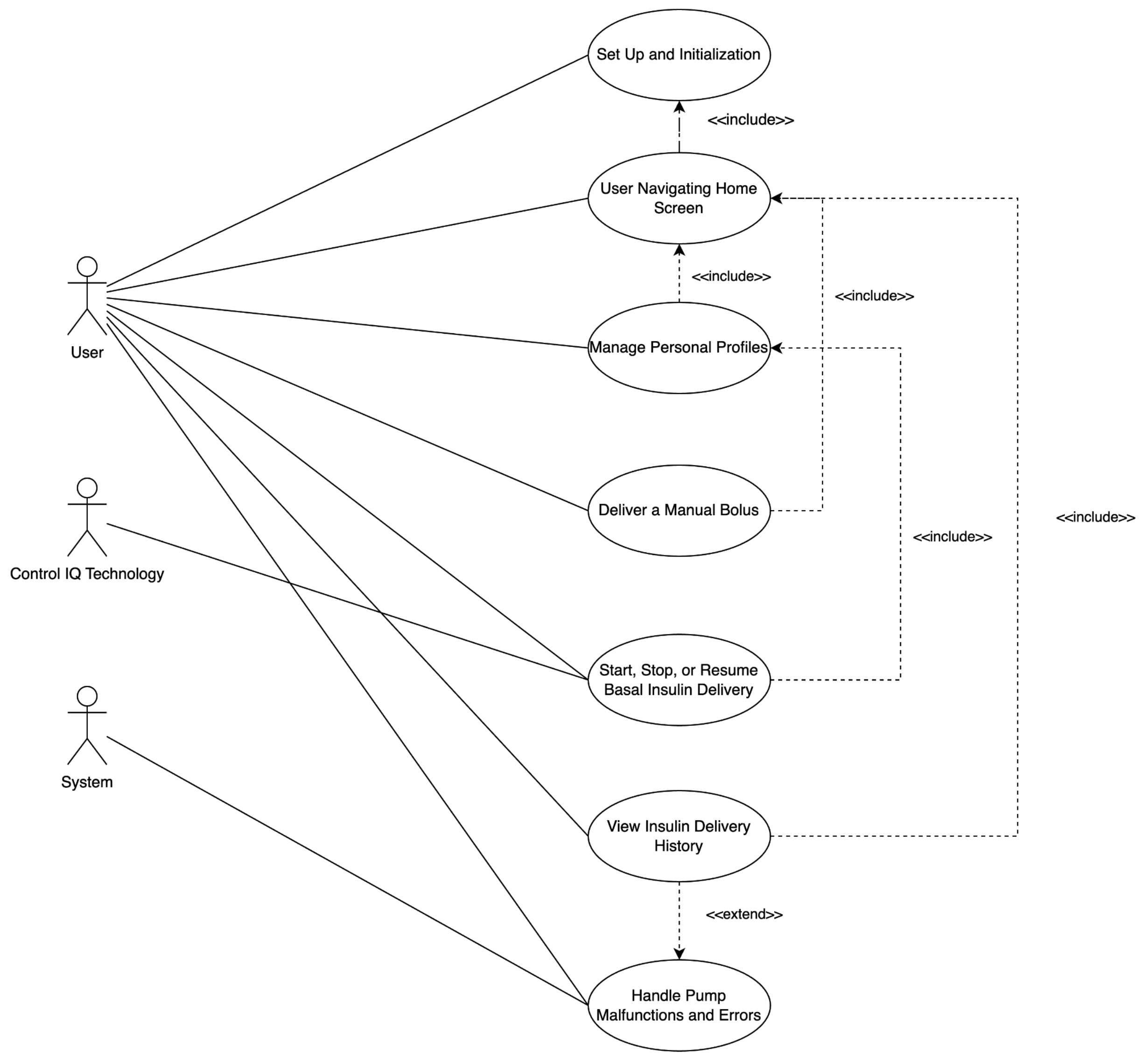
4. User Takes Corrective Action: The user follows the on-screen guidance to resolve the issue:
    a. Low Battery: User presses the "Charge Battery" button to recharge.
    b. Out of Battery: User presses the "Charge Battery" button to recharge.
    c. Low Insulin: User presses the "Refill Cartridge" button to refill the cartridge.
    d. Device Disconnection: User presses the "Reconnect Device" button to re-establish the connection.
    e. Occlusion: User presses the "Resolve Occlusion" button to clear the blockage.
    f. Hypoglycemia/Hyperglycemia: Insulin delivery is automatically calculated to fix the condition.
5. Resume Normal Operation: Once the issue is resolved, the system:
    a. Resumes insulin delivery at the previously set basal rate if safe to do so.

<u>Extensions:</u>

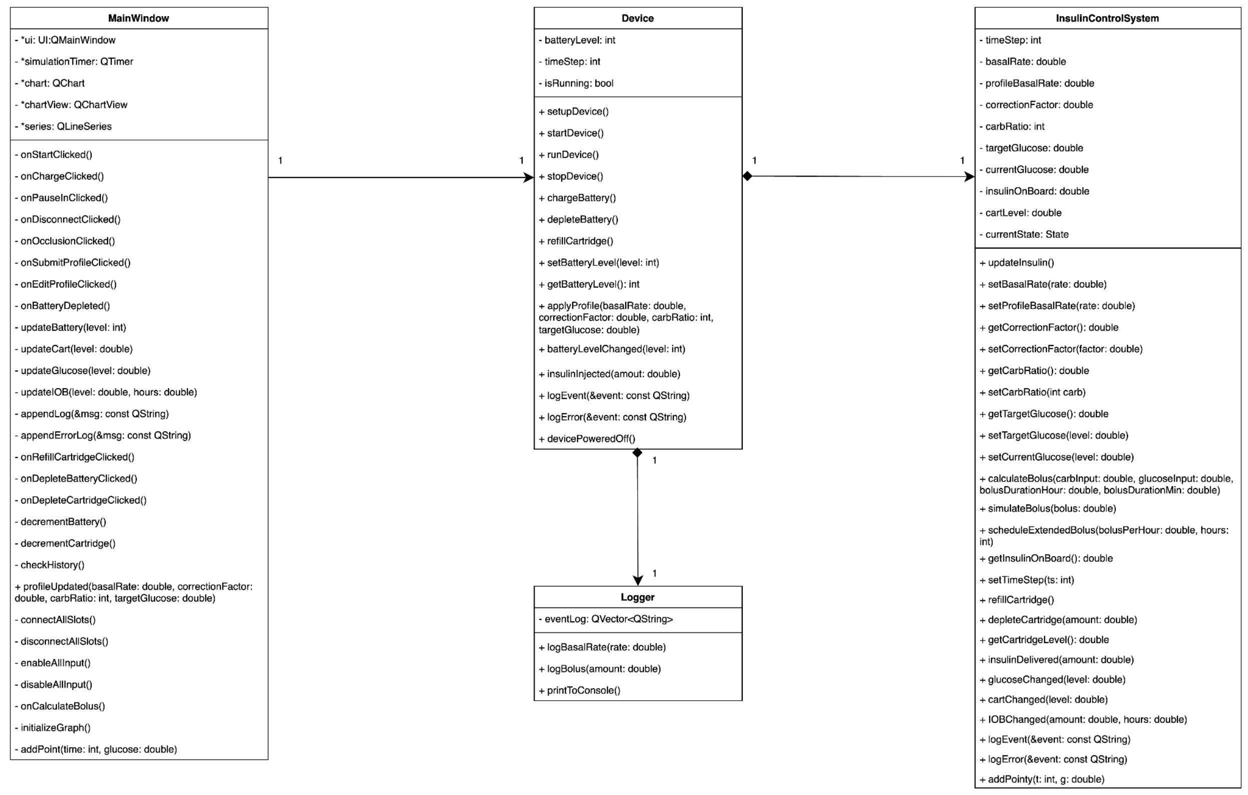
2.a. User Ignores Low Battery Warning:

- 2.a.1: If the battery level drops to 0%, the pump shuts down automatically, logs the event ("System powered off due to battery depletion."), and disables all controls except the charge button.
- 2.a.2: Upon recharging, the user must restart the pump by pressing "Power On."

Use Case Diagram:


Set Up and Initialization
<<include>>
User Navigating Home Screen
<<include>>
<<include>>
Manage Personal Profiles
User
Deliver a Manual Bolus
<<include>>
<<include>>
Control IQ Technology
Start, Stop, or Resume Basal Insulin Delivery
System
View Insulin Delivery History
<<extend>>
Handle Pump Malfunctions and Errors

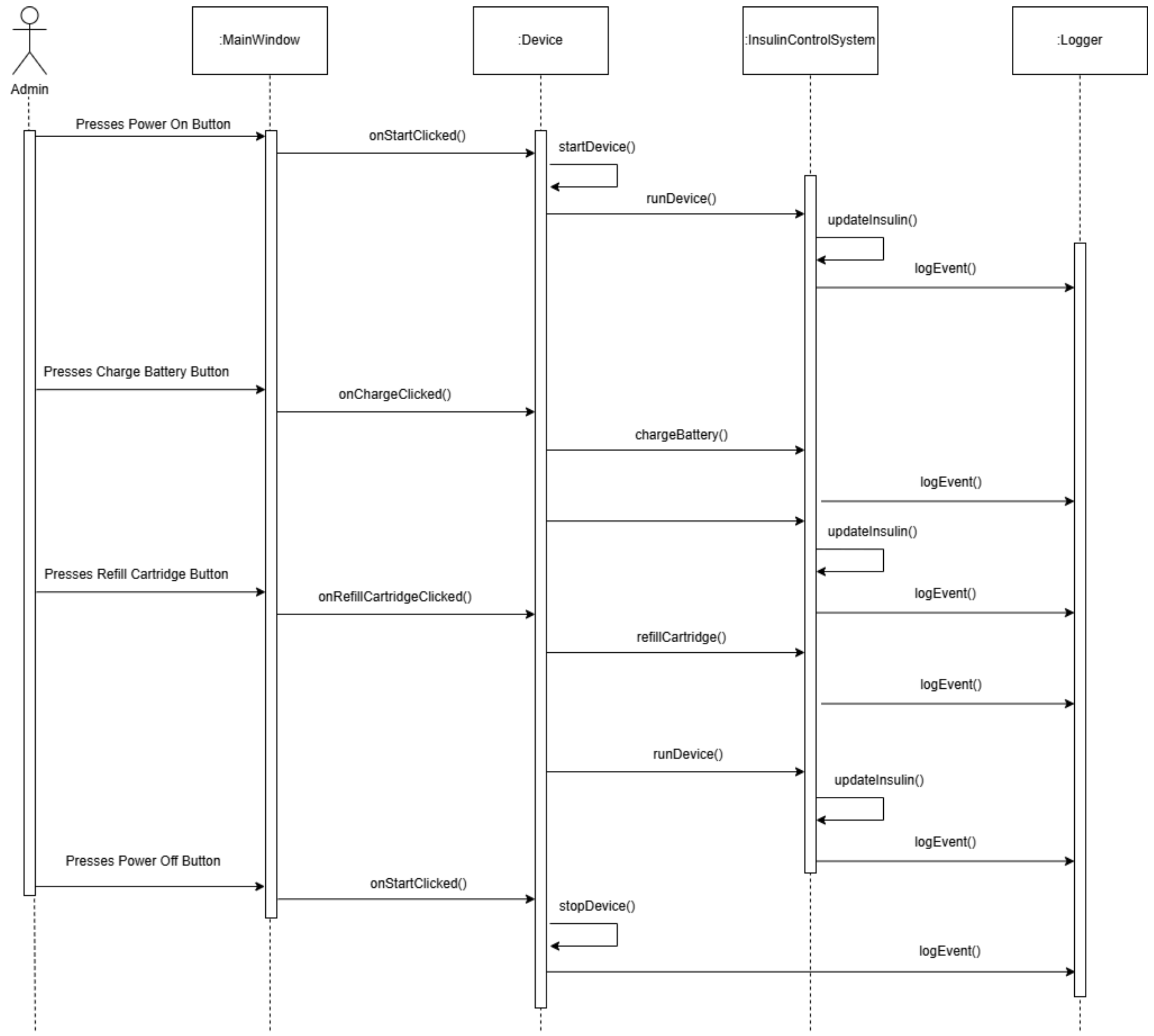
UML Class Diagram:


MainWindow
- *ui: UI:QMainWindow
- *simulationTimer: QTimer
- *chart: QChart
- *chartView: QChartView
- *series: QLineSeries
- onStartClicked()
- onChargeClicked()
- onPauseInClicked()
- onDisconnectClicked()
- onOcclusionClicked()
- onSubmitProfileClicked()
- onEditProfileClicked()
- onBatteryDepleted()
- updateBattery(level: int)
- updateCart(level: double)
- updateGlucose(level: double)
- updateIOB(level: double, hours: double)
- appendLog(&msg: const QString)
- appendErrorLog(&msg: const QString)
- onRefillCartridgeClicked()
- onDepleteBatteryClicked()
- onDepleteCartridgeClicked()
- decrementBattery()
- decrementCartridge()
- checkHistory()
+ profileUpdated(basalRate: double, correctionFactor: double, carbRatio: int, targetGlucose: double)
- connectAllSlots()
- disconnectAllSlots()
- enableAllInput()
- disableAllInput()
- onCalculateBolus()
- initializeGraph()
- addPoint(time: int, glucose: double)
1
1
Device
- batteryLevel: int
- timeStep: int
- isRunning: bool
+ setupDevice()
+ startDevice()
+ runDevice()
+ stopDevice()
+ chargeBattery()
+ depleteBattery()
+ refillCartridge()
+ setBatteryLevel(level: int)
+ getBatteryLevel(): int
+ applyProfile(basalRate: double, correctionFactor: double, carbRatio: int, targetGlucose: double)
+ batteryLevelChanged(level: int)
+ insulinInjected(amout: double)
+ logEvent(&event: const QString)
+ logError(&event: const QString)
+ devicePoweredOff()
1
1
InsulinControlSystem
- timeStep: int
- basalRate: double
- profileBasalRate: double
- correctionFactor: double
- carbRatio: int
- targetGlucose: double
- currentGlucose: double
- insulinOnBoard: double
- cartLevel: double
- currentState: State
+ updateInsulin()
+ setBasalRate(rate: double)
+ setProfileBasalRate(rate: double)
+ getCorrectionFactor(): double
+ setCorrectionFactor(factor: double)
+ getCarbRatio(): double
+ setCarbRatio(int carb)
+ getTargetGlucose(): double
+ setTargetGlucose(level: double)
+ setCurrentGlucose(level: double)
+ calculateBolus(carbInput: double, glucoseInput: double, bolusDurationHour: double, bolusDurationMin: double)
+ simulateBolus(bolus: double)
+ scheduleExtendedBolus(bolusPerHour: double, hours: int)
+ getInsulinOnBoard(): double
+ setTimeStep(ts: int)
+ refillCartridge()
+ depleteCartridge(amount: double)
+ getCartridgeLevel(): double
+ insulinDelivered(amount: double)
+ glucoseChanged(level: double)
+ cartChanged(level: double)
+ IOBChanged(amount: double, hours: double)
+ logEvent(&event: const QString)
+ logError(&event: const QString)
+ addPointy(t: int, g: double)
1
1
Logger
- eventLog: QVector<QString>
+ logBasalRate(rate: double)
+ logBolus(amount: double)
+ printToConsole()

UML Sequence Diagrams:

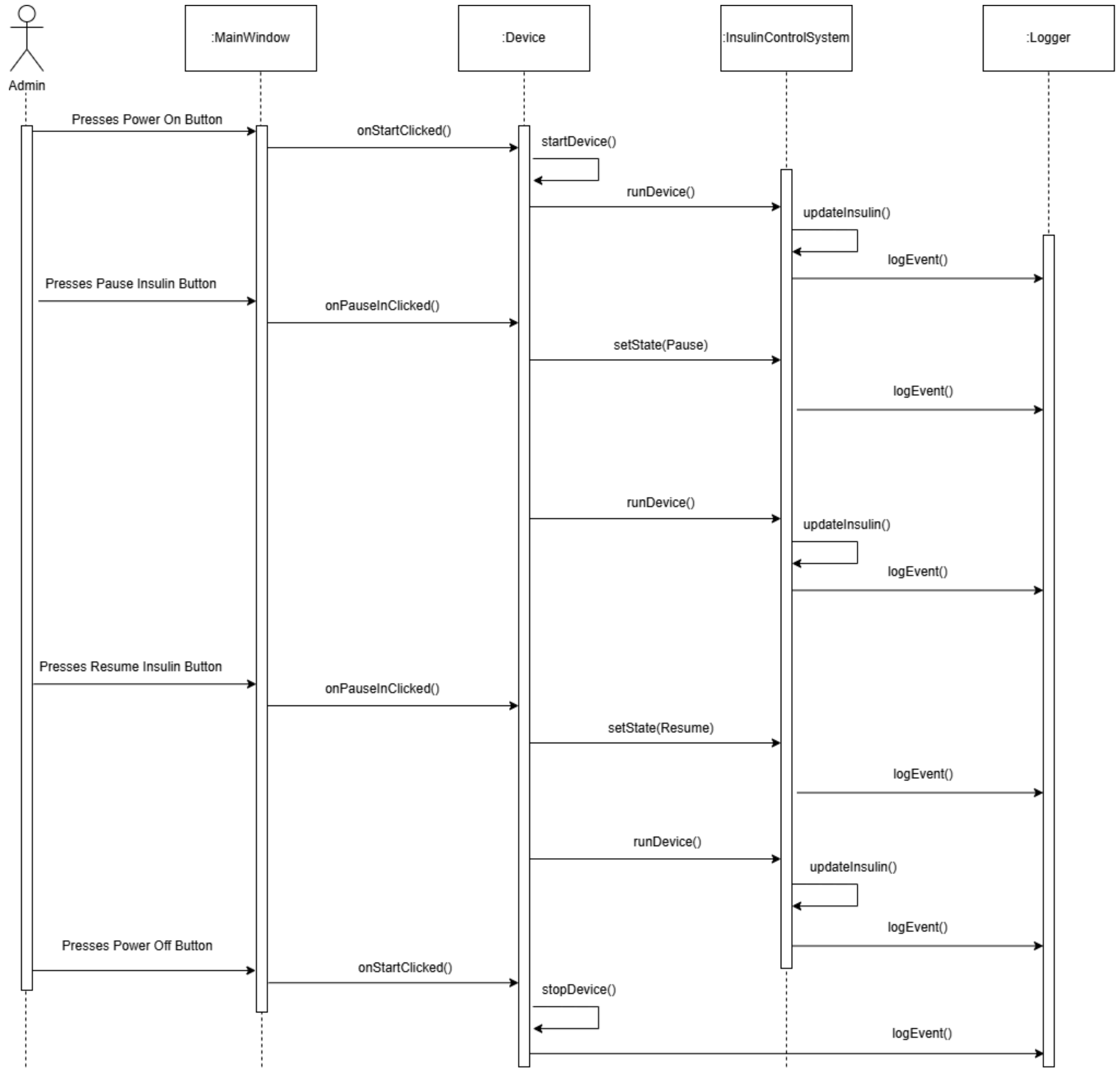
Normal Sequence 1:

The device is powered on, and the admin charges the battery and refills the cartridge. Then, the device is powered off.



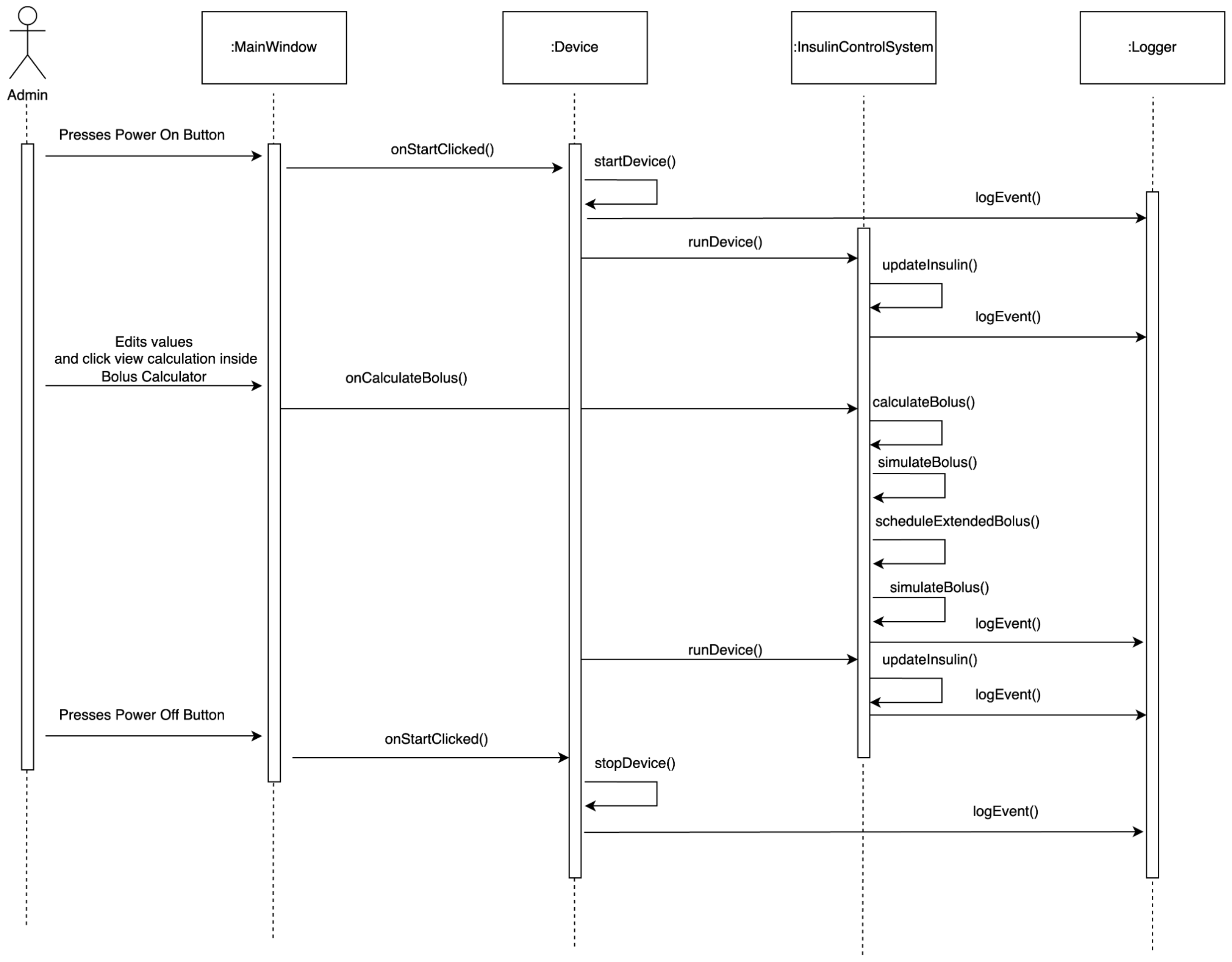
Normal Sequence 2:

The device is powered on, and the admin pauses and resumes insulin. Then, the device is powered off.



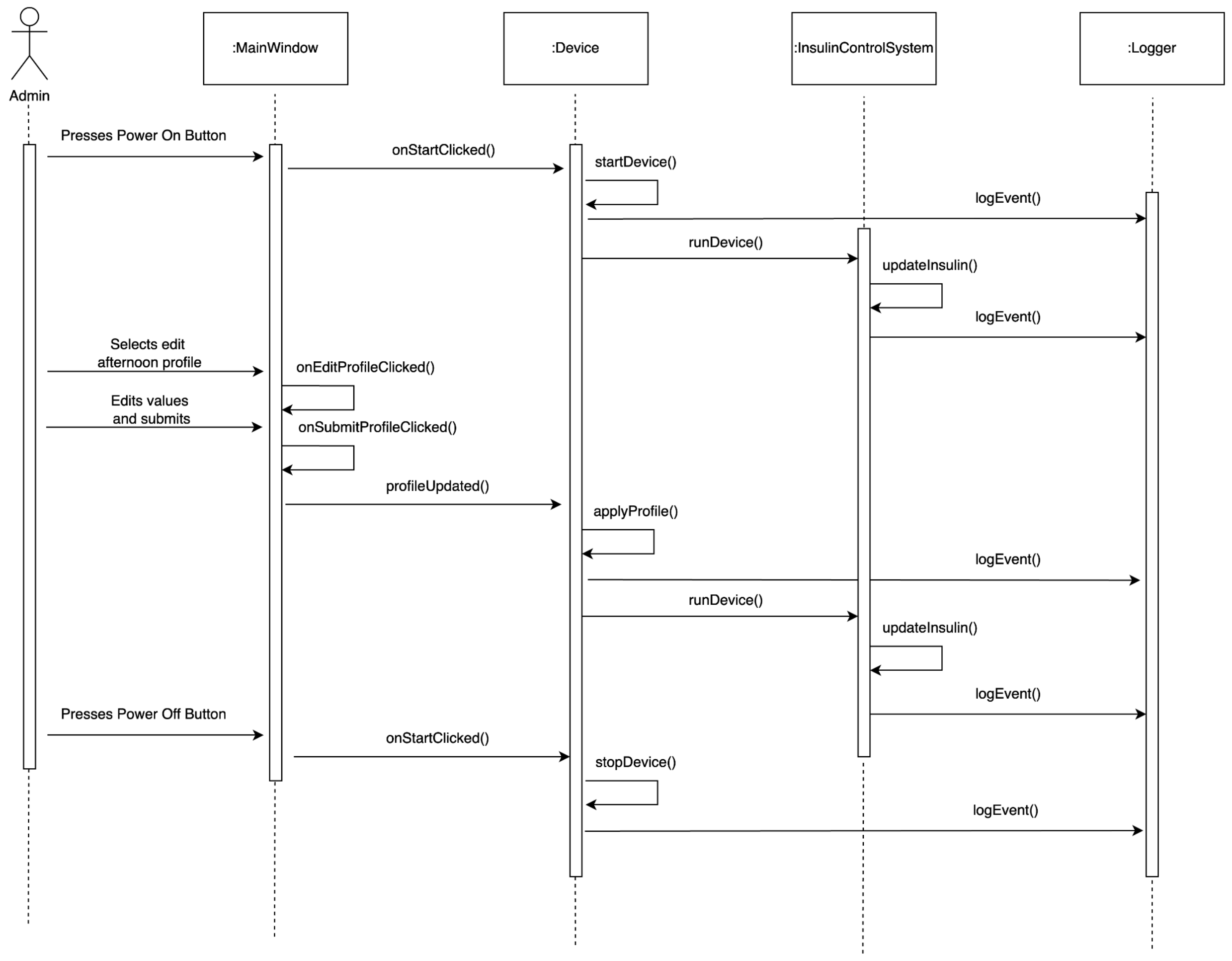
Normal Sequence 3:

The device is powered on, and the admin does a bolus injection (Carbs: 45g | Glucose 4.5mmol/L | Duration 3hr). Then, the device is powered off.



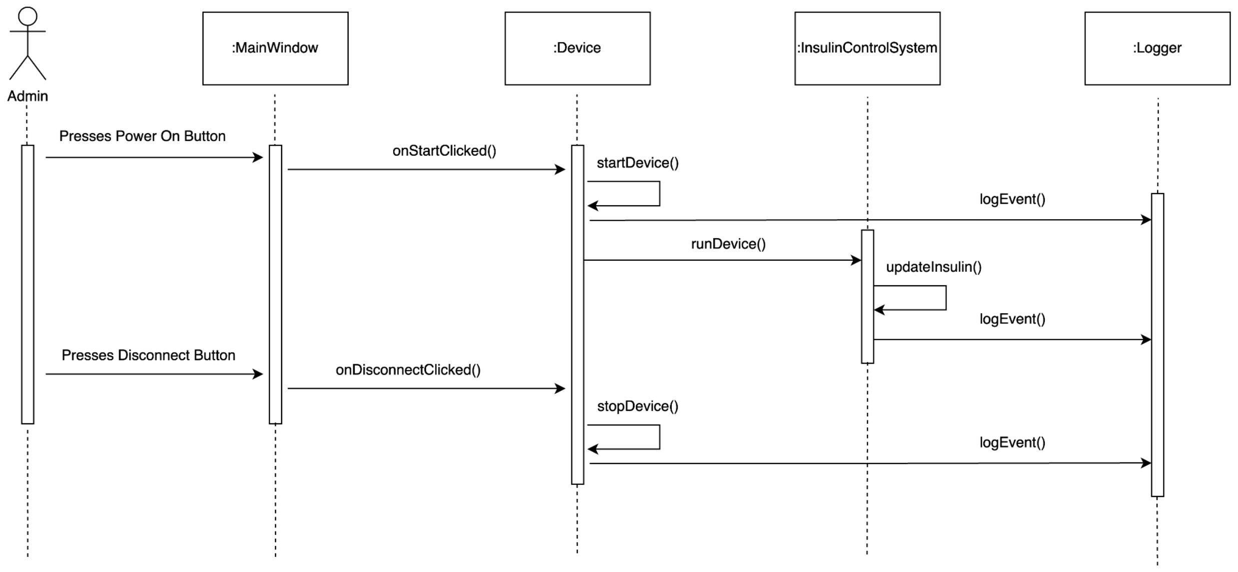
Normal Sequence 4:

The device is powered on (defaults to morning personal profile at first), the admin edits the afternoon personal profile (Basal Rate: 1.947u/1hr | Correction Factor: 2.8 1u:mmol/1hr | Carb Ratio: 13 1u:g | Blood Glucose: 5.7mmol/L) and applies it. Then, the device is powered off.



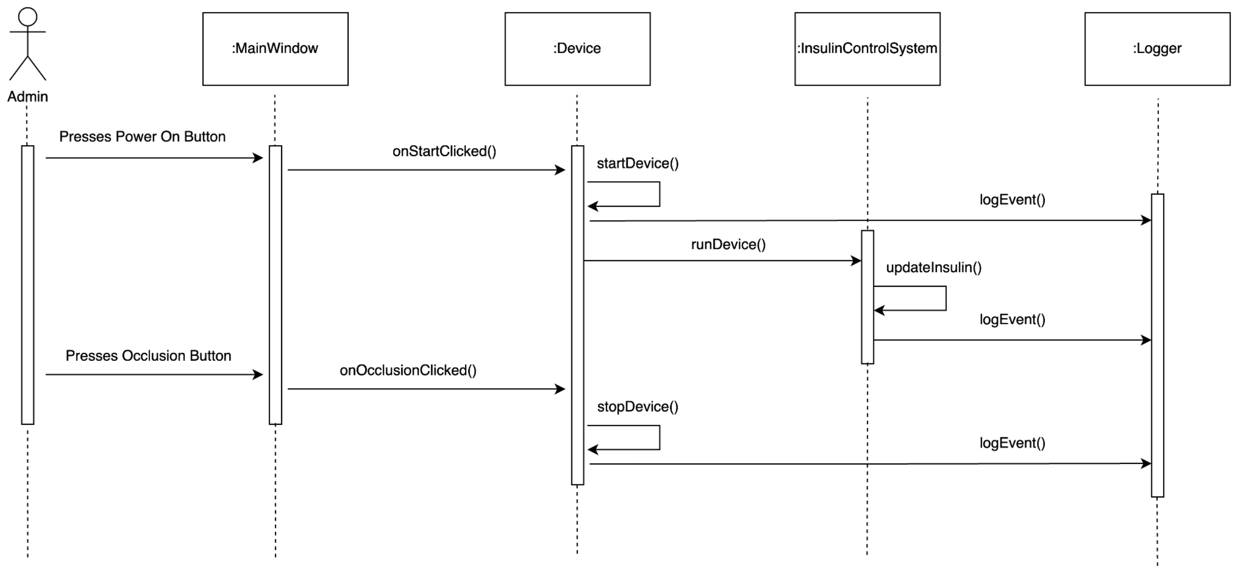
Safety Sequence 1:

The device is powered on, then the device gets disconnected from the user's body.



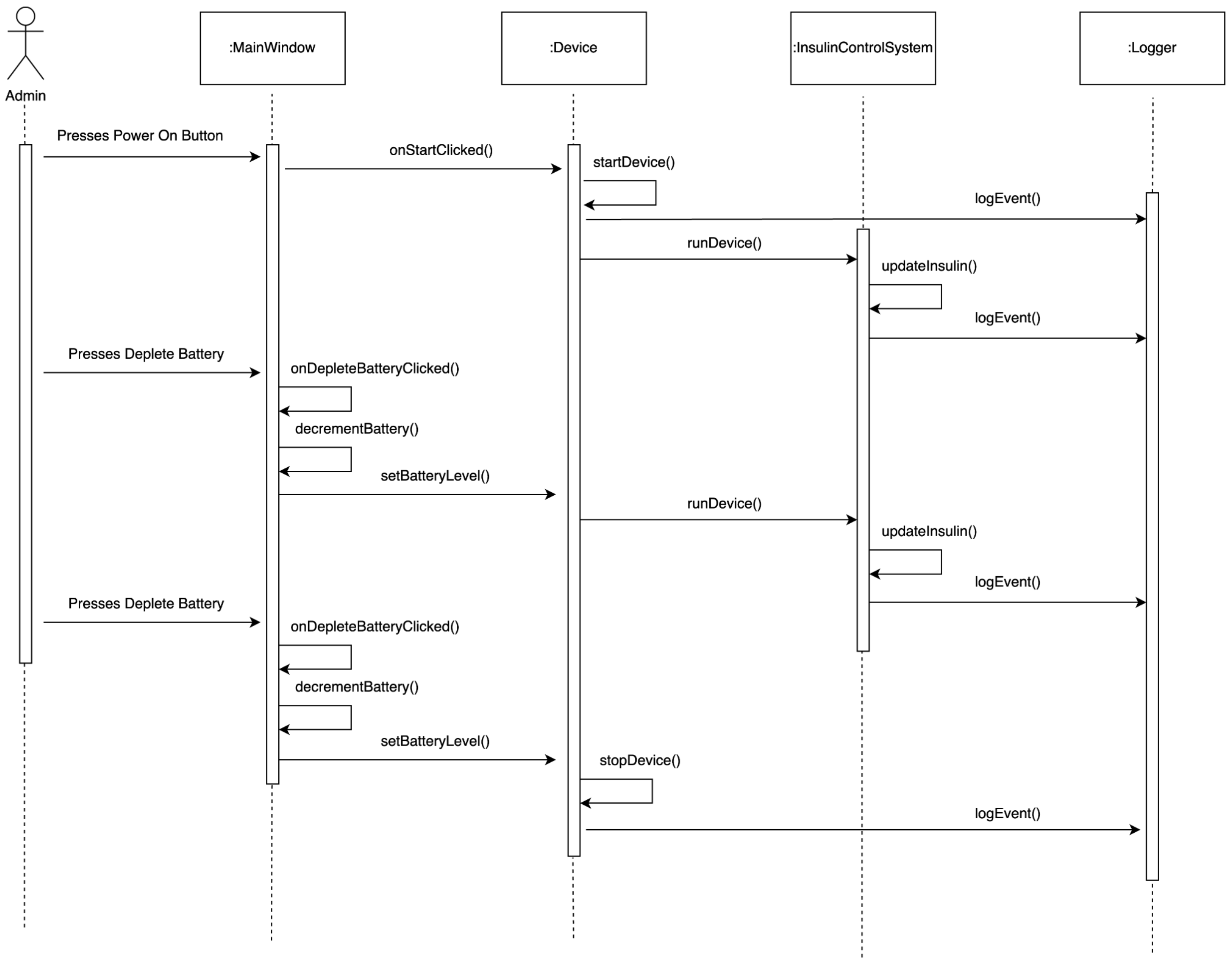
Safety Sequence 2:

The device is powered on, and then a blockage/occlusion occurs.

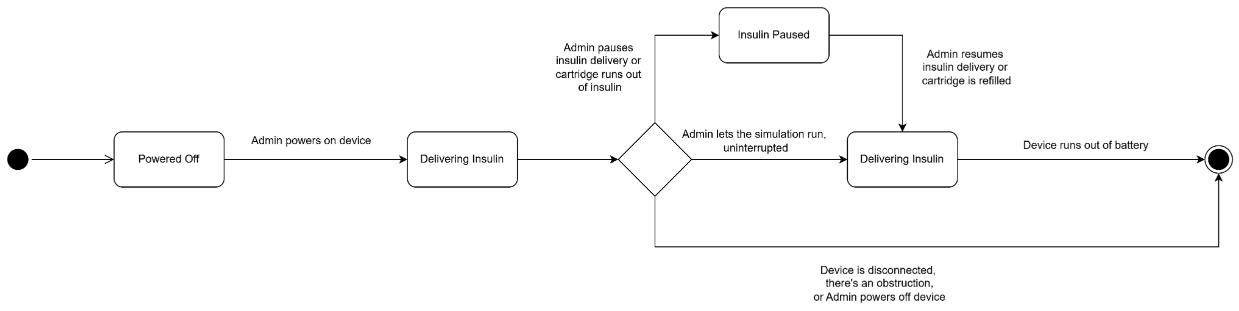


Safety Sequence 3:

The device is powered on, then the device runs out of battery and shuts down.



UML State Machine Diagram:


Powered Off
Admin powers on device
Delivering Insulin
Admin pauses insulin delivery or cartridge runs out of insulin
Insulin Paused
Admin resumes insulin delivery or cartridge is refilled
Admin lets the simulation run, uninterrupted
Delivering Insulin
Device runs out of battery
Device is disconnected, there's an obstruction, or Admin powers off device

Textual Explanation:

This design prioritizes core functionality based on the project specification, with simplifications made according to guidance from the professor. First, the application uses three predefined profiles instead of full create/delete functionality to simplify interaction while still supporting flexible insulin delivery settings. Next, the interface is limited to a single screen, making home and options buttons unnecessary. Also, predictions are shown in the logs rather than the graph to reduce UI clutter. Additionally, bolus calculation uses default glucose values, and there is no manual override, as dosing is automatically determined by the app logic in line with Control IQ behavior. A PIN screen and icons were also omitted, as the professor confirmed they were not required. Furthermore, logging replaces visual indicators like icons by clearly communicating system feedback, including profile changes, bolus events, and warnings. Finally, to support consistent time-based behavior across features, simulation time is implemented such that one time step corresponds to one real-time second.

Qt's observer pattern (signal-slot) is used throughout to decouple simulation logic from UI controls. A logger pattern centralizes output and provides traceability across the system. While not implemented as a formal state pattern, internal state tracking is used to control behavior such as basal suspension and delivery resumption. These design choices support a clean, modular, and testable architecture while staying aligned with both project requirements and practical constraints. Moreover, Unit tests were written to validate key functionality, including bolus calculation, profile switching, and logging, helping ensure reliability and correctness.

Video:

https://www.youtube.com/watch?v=DZweDLuIvL4

Requirements Traceability Matrix:

ID	Requirement	Use Case	Design Element	Implementation Elements	Test
REQ-1	The pump must have a power on and off functionality.	Use Case 1: Set up and Initialization	Device::startDevice(), Device::stopDevice()	insulinpump.cpp: Device::startDevice(), Device::stopDevice()	testDeviceStartStop()
REQ-2	The home screen must display battery life, date & time, IOB, status, insulin level, Tandem logo, and CGM data graph.	Use Case 2: User Navigating Home screen	MainWindow UI elements	mainwindow.cpp: MainWindow constructor, update functions	Manual Test
REQ-3	The home screen must have buttons for bolus calculator, options, and returning to the home screen.	Use Case 2: User Navigating Home screen	MainWindow UI buttons	mainwindow.ui, mainwindow.cpp: button slots	Manual Test
REQ-4	The pump must deliver basal insulin continuously unless interrupted.	Use Case 5: Start, Stop, or Resume Basal Insulin Delivery	InsulinControlSystem::updateInsulin()	insulinpump.cpp: InsulinControlSystem::updateInsulin()	testBasalRateSettings()
REQ-5	Control-IQ must adjust basal insulin based on CGM predictions.	Use Case 5: Start, Stop, or Resume Basal Insulin Delivery	InsulinControlSystem::updateInsulin()	insulinpump.cpp: InsulinControlSystem::updateInsulin()	testGlucoseManagement()
REQ-6	Insulin delivery events must be logged.	Use Case 5: Start, Stop, or Resume Basal Insulin Delivery	logEvent signal, appendLog slot	insulinpump.cpp: emit logEvent, mainwindow.cpp: appendLog	Manual Test
REQ-7	Users must be able to access the bolus calculator.	Use Case 4: Deliver a Manual Bolus	MainWindow::onCalculateBolus()	mainwindow.cpp: onCalculateBolus()	testInsulinBolus()
REQ-8	The bolus calculator must suggest a dose based on inputs and settings.	Use Case 4: Deliver a Manual Bolus	MainWindow::onCalculateBolus(), InsulinControlSystem::calculateBolus()	mainwindow.cpp: onCalculateBolus(), insulinpump.cpp: calculateBolus()	testInsulinBolus()
REQ-9	Users must be able to override the suggested bolus dose.	Use Case 4: Deliver a Manual Bolus	MainWindow::onCalculateBolus()	mainwindow.cpp: onCalculateBolus(), insulinpump.cpp: calculateBolus()	Manual Test
REQ-10	Bolus delivery must be able	Use Case 4:	InsulinControlSystem	insulinpump.cpp:	Manual Test

	to be canceled or stopped.	Deliver a Manual Bolus	::setState()	InsulinControlSystem::setState(), scheduleExtendedBolus()	
REQ-11	Users must be able to create, read, update, and delete personal profiles.	Use Case 3: Manage Personal Profiles (CRUD)	MainWindow UI for profiles	mainwindow.ui, mainwindow.cpp: profile-related slots	Manual Test
REQ-12	Personal profiles must include basal rates, carb ratios, correction factors, and target glucose levels.	Use Case 3: Manage Personal Profiles (CRUD)	InsulinControlSystem profile settings	insulinpump.cpp: InsulinControlSystem setters	testProfiles()
REQ-13	One personal profile must be set as active.	Use Case 3: Manage Personal Profiles (CRUD)	MainWindow::onSubmitProfileClicked()	mainwindow.cpp: onSubmitProfileClicked()	testProfiles()
REQ-14	Users must be able to view a log of insulin delivery events.	Use Case 6: View Insulin Delivery History	MainWindow::appendLog()	mainwindow.cpp: appendLog()	Manual Test
REQ-15	The system must detect and alert for low battery.	Use Case 7: Handle Pump Malfunctions and Errors	Device::setBatteryLevel()	insulinpump.cpp: Device::setBatteryLevel()	testBatteryManagement()
REQ-16	The system must detect and alert for low insulin.	Use Case 7: Handle Pump Malfunctions and Errors	InsulinControlSystem::depleteCartridge()	insulinpump.cpp: InsulinControlSystem::depleteCartridge()	testCartridgeLevel()
REQ-17	The system must detect and alert for CGM disconnection.	Use Case 7: Handle Pump Malfunctions and Errors	MainWindow::onDisconnectClicked()	mainwindow.cpp: onDisconnectClicked()	Manual Test
REQ-18	The system must detect and alert for occlusion.	Use Case 7: Handle Pump Malfunctions and Errors	MainWindow::onOcclusionClicked()	mainwindow.cpp: onOcclusionClicked()	Manual Test
REQ-19	The system must detect and alert for pump shutdown.	Use Case 7: Handle Pump Malfunctions and Errors	Device::stopDevice()	insulinpump.cpp: Device::stopDevice()	Manual Test
REQ-20	Insulin delivery must be	Use Case 7:	InsulinControlSystem	insulinpump.cpp:	Manual Test

	suspended in critical errors.	Handle Pump Malfunctions and Errors	::setState()	InsulinControlSystem:: setState()	
REQ-21	Errors must be logged.	Use Case 7: Handle Pump Malfunctions and Errors	logEvent signal, appendLog slot	insulinpump.cpp: emit logEvent, mainwindow.cpp: appendLog	Manual Test