Tandem t:slim X2 Insulin Pump Simulation

Final Assignment

Object Oriented Software Engineering (COMP 3004)

Carleton University

By

Hollen Lo (101260373), Ariz Kazani (101311311), Kassem Taha (101304020) and Niharika Ramireddy (101285226)

April 2025

Final Project Specification

Deliverables:

Use Cases

Use Case: Simulate Insulin Pump Operations

Primary Actor: Patient (User), Healthcare Provider (Administrator)

Scope: t:slim X2 Insulin Pump Simulator

Level: User goal

Stakeholders and Interests:

Patient: Wants to manage insulin delivery, monitor glucose levels, and ensure proper diabetes management.

Healthcare Provider: Wants to monitor patient data, adjust settings, and ensure the pump operates correctly

System: Wants to process data, deliver insulin accurately, and handle errors or malfunctions.

Precondition: The simulation is running, and the pump is powered on.

Minimal Guarantee: The system logs all user interactions, insulin delivery events, and errors for review.

Success Guarantee: The patient successfully manages insulin delivery, monitors glucose levels, and receives actionable insights. The system updates all data and states accurately.

Main Success Scenario:

1.Patient powers on the pump and unlocks it.

- The patient holds the power button to turn on the pump.
- The system logs the power-on event and updates the pump's state to "Operational."

2.Patient navigates the home screen.

- The patient views the home screen, which displays:

- Battery indicator (upper-left corner).
- Insulin fill gauge (upper-right corner).
- "Insulin on Board" (IOB) indicator.
- Navigation buttons (e.g., bolus button, options button).

· The system logs the interaction and updates the display.

3.Patient sets up or updates a personal profile.

· The patient navigates to the personal profiles section.

· The patient creates a new profile or updates an existing one by entering:

- Basal rates.
- Carbohydrate ratios.
- Correction factors.
- Target glucose levels.

· The system saves the profile and logs the changes.

4.Patient delivers a manual bolus.

· The patient accesses the bolus calculator via the home screen or bolus button.

· The patient enters current blood glucose level and carbohydrate intake (manually or via CGM).

· The pump suggests a bolus dose based on programmed settings.

· The patient confirms or adjusts the dose.

· The system delivers the bolus and logs the event.

5.Patient starts, stops, or resumes basal insulin delivery.

· The patient selects a basal rate from the active personal profile.

· The pump starts delivering insulin at the specified rate.

· The patient manually stops insulin delivery if needed.

· The pump automatically stops insulin delivery if Control IQ detects low glucose levels (< 3.9 mmol/L).

- The pump automatically provides a fixed insulin delivery if Control IQ detects high glucose levels (> 10 mmol/L).
- The patient resumes insulin delivery when glucose levels stabilize.
- The system logs all changes in insulin delivery.

6.Patient reviews pump information and history.

- The patient navigates to the history section.
- The patient views logs of:
  - Basal rates.
  - Bolus injections.
  - User/Machine actions.
  - Safety events
- The system displays the data and updates the logs.

7.Patient handles pump malfunctions.

- The pump detects a malfunction (e.g., low battery, low insulin, occlusion).
- The pump triggers an alert and suggests corrective actions.
- The patient takes corrective action (e.g., recharges battery, checks infusion site).
- The system logs the malfunction and updates the pump's state.

8.Patient visualizes data and generates reports.

- The patient navigates to the data visualization section.
- The patient selects a time range (within 24 hours).
- The system generates graphs and reports based on the selected data.
- The patient analyzes the data for actionable insights.

9.Patient interacts with Control IQ technology.

- Control IQ reads real-time CGM data.
- Control IQ adjusts basal insulin delivery based on glucose trends.
- The patient reviews the adjustments made by Control IQ.
- The system logs all adjustments and updates the pump's state.

10.System collects and processes data for analysis.

- The system collects data from:
  - Insulin delivery events.
  - Safety events.
  - User interactions.
- The system processes the data to identify patterns and trends.
- The system stores the processed data for visualization and reporting.

Extensions

1. Low Power Warning

    1A. System detects the low battery level.

    1B. Logs console warning.

2. Power Failure (No Power)

    2A. System detects critical power loss.

    2B. Immediately terminates simulation.

3.Invalid profile settings:

    3A. Duplicate/case-sensitive name: Profile ignored (not registered).

    3B. Invalid time period: System auto-switches to next valid profile (if exists).

4. Low Insulin Warning

    4A. System detects the insulin level below required.

    4B. Logs console warning; continues normal insulin delivery operations

5. Pump runs out of insulin:

    5A. The system detects no insulin and triggers an alert.

    5B. The patient replaces the insulin cartridge and resumes insulin delivery.

    5C. If no input from the user don't deliver the insulin.

6. CGM disconnects(not within time period):

    6A. The system detects the disconnection and notifies the patient.

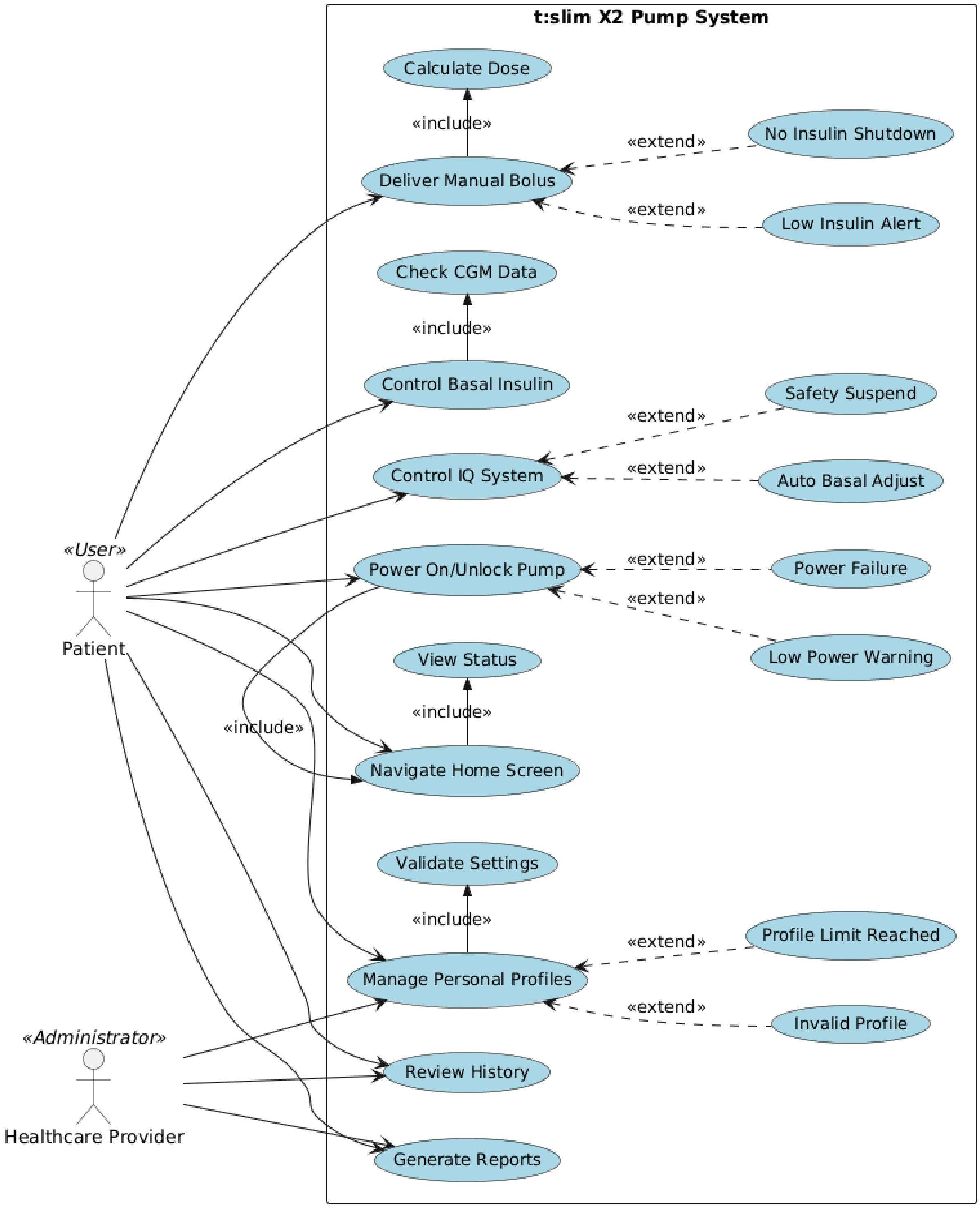
6B. Switch to the next possible profile if no profile is available suspends Control IQ.

7. Profile limit enforcement:

7A. 6 profiles active: New profiles ignored (no override).

7B. Invalid input (missing fields): Profile rejected (not registered).

Use Case Diagram:

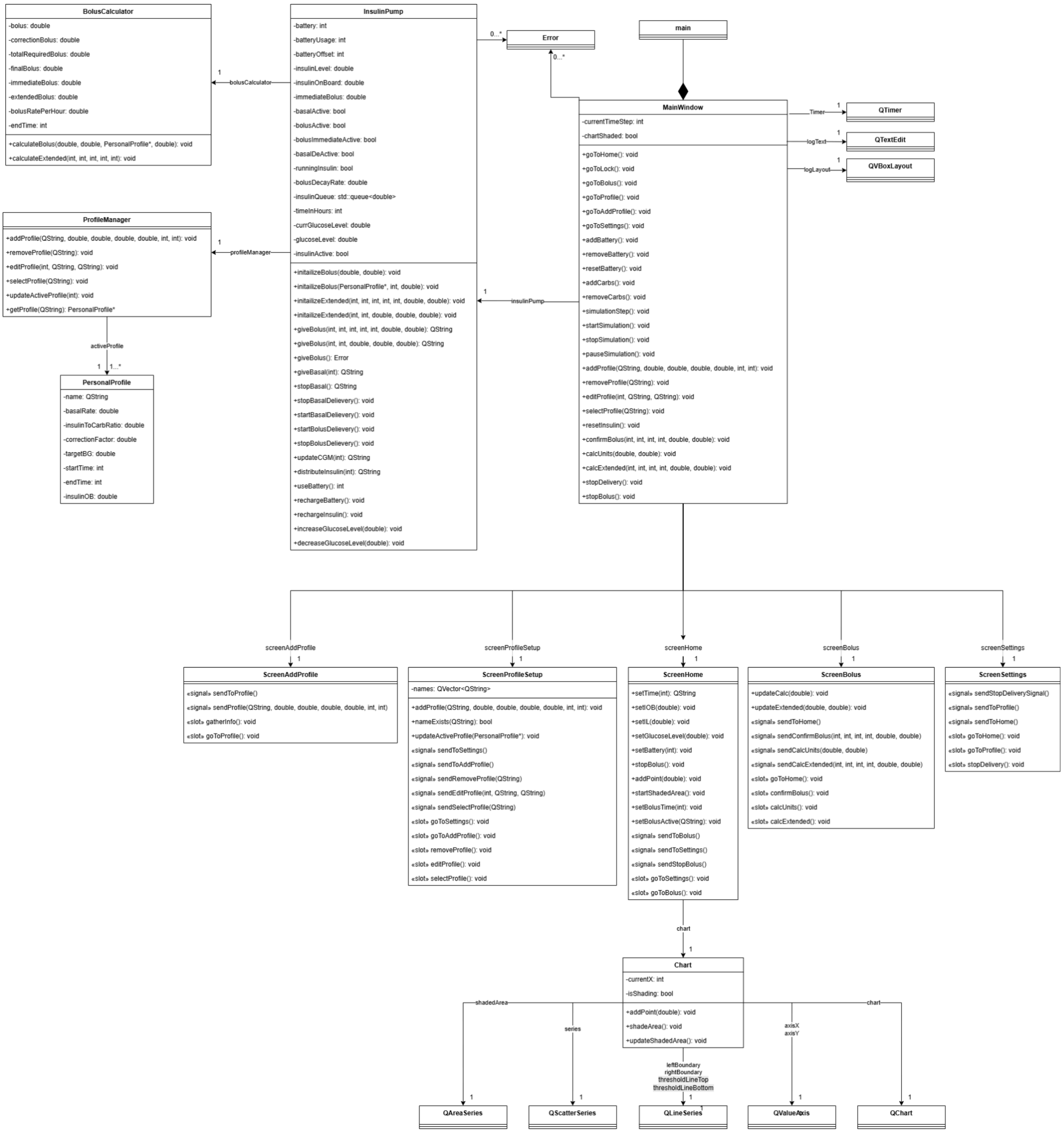


Design documentation – structure and behavior:

tUML Class diagram.

(Insulin Pump is our Controller class and will connect with the UI and other classes)
(we will use the notation of <signal> to show functions under signals: and <slots> to show functions under private slots.)

Diagram:



BolusCalculator
-bolus: double
-correctionBolus: double
-totalRequiredBolus: double
-finalBolus: double
-immediateBolus: double
-extendedBolus: double
-bolusRatePerHour: double
-endTime: int
+calculateBolus(double, double, PersonalProfile*, double): void
+calculateExtended(int, int, int, int, int): void
1
bolusCalculator
InsulinPump
-battery: int
-batteryUsage: int
-batteryOffset: int
-insulinLevel: double
-insulinOnBoard: double
-immediateBolus: double
-basalActive: bool
-bolusActive: bool
-bolusImmediateActive: bool
-basalDeActive: bool
-runningInsulin: bool
-bolusDecayRate: double
-insulinQueue: std::queue<double>
-timeInHours: int
-currGlucoseLevel: double
-glucoseLevel: double
-insulinActive: bool
+initailizeBolus(double, double): void
+initailizeBolus(PersonalProfile*, int, double): void
+initailizeExtended(int, int, int, int, int, double, double): void
+initailizeExtended(int, int, double, double, double): void
+giveBolus(int, int, int, int, int, double, double): QString
+giveBolus(int, int, double, double, double): QString
+giveBolus(): Error
+giveBasal(int): QString
+stopBasal(): QString
+stopBasalDelievery(): void
+startBasalDelievery(): void
+startBolusDelievery(): void
+stopBolusDelievery(): void
+updateCGM(int): QString
+distributeInsulin(int): QString
+useBattery(): int
+rechargeBattery(): void
+rechargeInsulin(): void
+increaseGlucoseLevel(double): void
+decreaseGlucoseLevel(double): void
0...*
Error
0...*
main
MainWindow
-currentTimeStep: int
-chartShaded: bool
+goToHome(): void
+goToLock(): void
+goToBolus(): void
+goToProfile(): void
+goToAddProfile(): void
+goToSettings(): void
+addBattery(): void
+removeBattery(): void
+resetBattery(): void
+addCarbs(): void
+removeCarbs(): void
+simulationStep(): void
+startSimulation(): void
+stopSimulation(): void
+pauseSimulation(): void
+addProfile(QString, double, double, double, double, int, int): void
+removeProfile(QString): void
+editProfile(int, QString, QString): void
+selectProfile(QString): void
+resetInsulin(): void
+confirmBolus(int, int, int, int, double, double): void
+calcUnits(double, double): void
+calcExtended(int, int, int, int, double, double): void
+stopDelivery(): void
+stopBolus(): void
Timer
1
QTimer
logText
1
QTextEdit
logLayout
1
QVBoxLayout
1
insulinPump
ProfileManager
+addProfile(QString, double, double, double, double, int, int): void
+removeProfile(QString): void
+editProfile(int, QString, QString): void
+selectProfile(QString): void
+updateActiveProfile(int): void
+getProfile(QString): PersonalProfile*
1
profileManager
activeProfile
1
1...*
PersonalProfile
-name: QString
-basalRate: double
-insulinToCarbRatio: double
-correctionFactor: double
-targetBG: double
-startTime: int
-endTime: int
-insulinOB: double
screenAddProfile
1
ScreenAddProfile
«signal» sendToProfile()
«signal» sendProfile(QString, double, double, double, double, int, int)
«slot» gatherInfo(): void
«slot» goToProfile(): void
screenProfileSetup
1
ScreenProfileSetup
-names: QVector<QString>
+addProfile(QString, double, double, double, double, int, int): void
+nameExists(QString): bool
+updateActiveProfile(PersonalProfile*): void
«signal» sendToSettings()
«signal» sendToAddProfile()
«signal» sendRemoveProfile(QString)
«signal» sendEditProfile(int, QString, QString)
«signal» sendSelectProfile(QString)
«slot» goToSettings(): void
«slot» goToAddProfile(): void
«slot» removeProfile(): void
«slot» editProfile(): void
«slot» selectProfile(): void
screenHome
1
ScreenHome
+setTime(int): QString
+setIOB(double): void
+setIL(double): void
+setGlucoseLevel(double): void
+setBattery(int): void
+stopBolus(): void
+addPoint(double): void
+startShadedArea(): void
+setBolusTime(int): void
+setBolusActive(QString): void
«signal» sendToBolus()
«signal» sendToSettings()
«signal» sendStopBolus()
«slot» goToSettings(): void
«slot» goToBolus(): void
screenBolus
1
ScreenBolus
+updateCalc(double): void
+updateExtended(double, double): void
«signal» sendToHome()
«signal» sendConfirmBolus(int, int, int, int, double, double)
«signal» sendCalcUnits(double, double)
«signal» sendCalcExtended(int, int, int, int, double, double)
«slot» goToHome(): void
«slot» confirmBolus(): void
«slot» calcUnits(): void
«slot» calcExtended(): void
screenSettings
1
ScreenSettings
«signal» sendStopDeliverySignal()
«signal» sendToProfile()
«signal» sendToHome()
«slot» goToHome(): void
«slot» goToProfile(): void
«slot» stopDelivery(): void
chart
1
Chart
-currentX: int
-isShading: bool
+addPoint(double): void
+shadeArea(): void
+updateShadedArea(): void
shadedArea
series
leftBoundary
rightBoundary
thresholdLineTop
thresholdLineBottom
axisX
axisY
chart
1
1
1
1
1
1
QAreaSeries
QScatterSeries
QLineSeries
QValueAxis
QChart

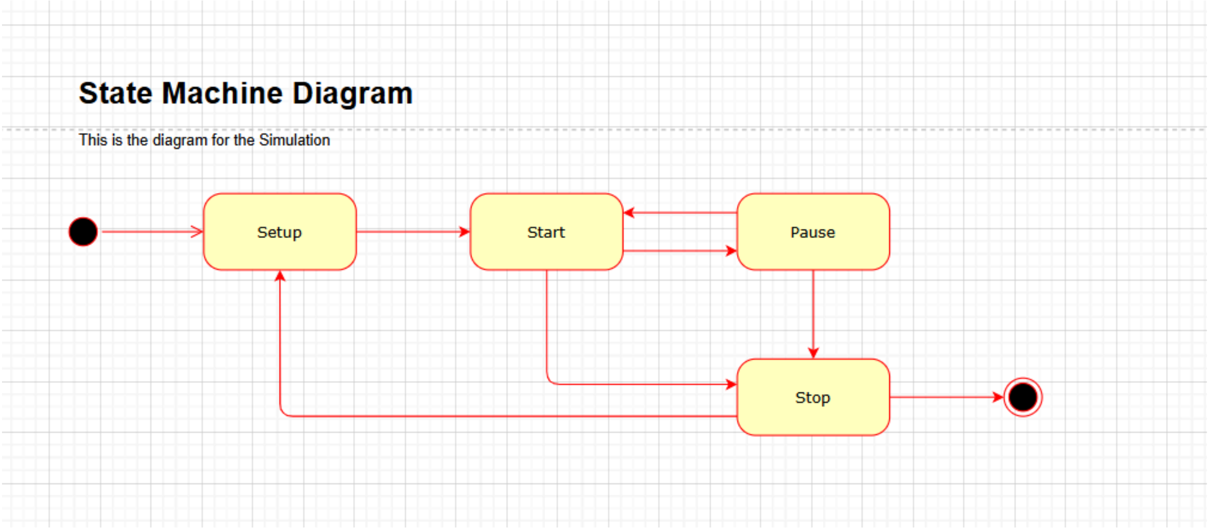
draw.io link:
https://drive.google.com/file/d/1-Qqzq0VAoRe8-rqx_ARm3A3vxupE5_de/view?usp=sharing

UML Sequence Diagrams

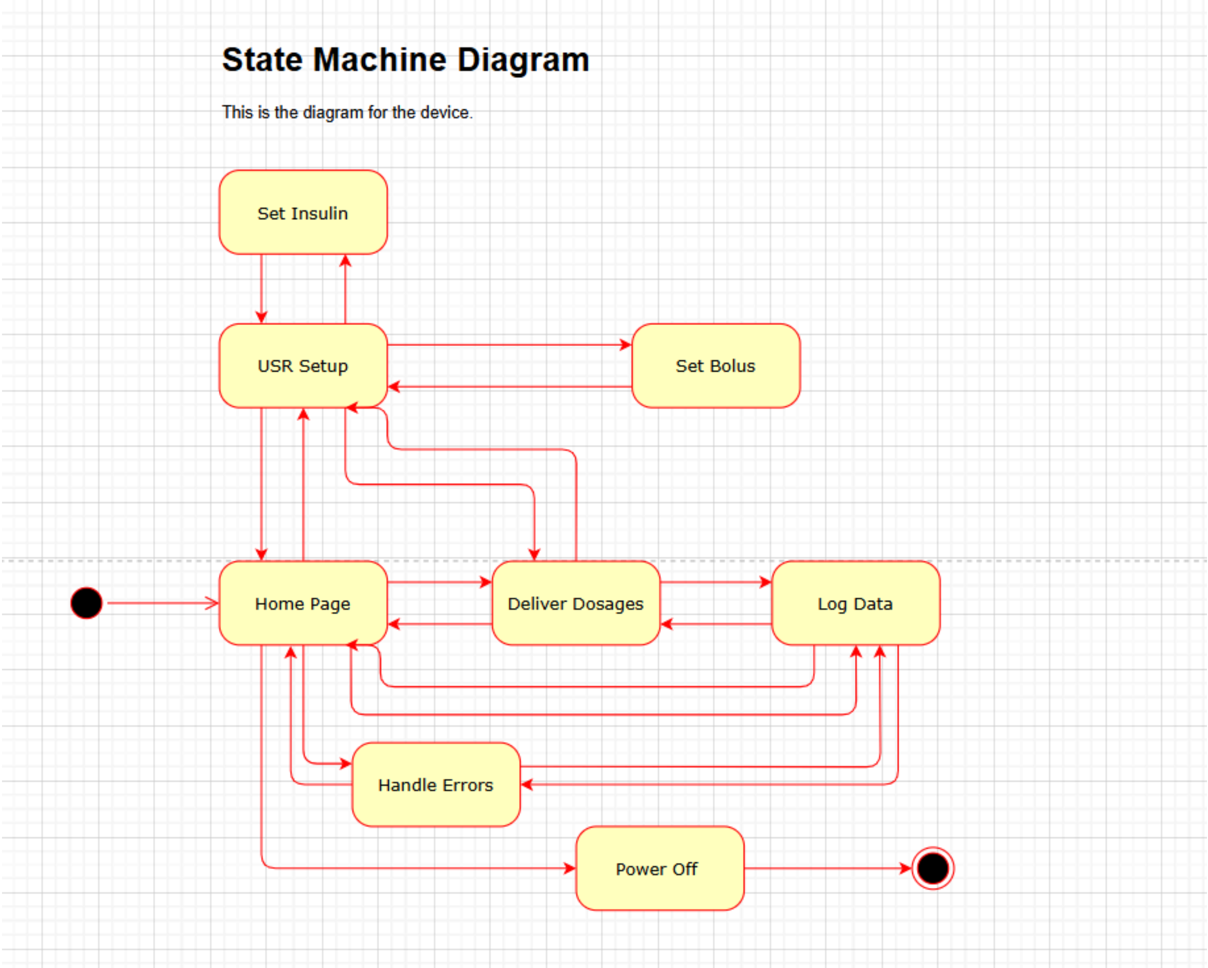
(scenarios match videos)
(certain actions within the video are not reflected as it would , for example testing the max number of profiles and deleting profiles were not included)
(not every timestep is documented, only significant events like bolus delivery, error events, switching profiles, etc are displayed, the function simulationStep(X) illustrates the time(in minutes) at which a significant event occurs with X, in actual code this parameter doesn't exist)
profile = X in profileManager is not a function but illustrates which profile is being switched to
Only a the high/low glucose level warnings between Morning and Afternoon profiles are are shown in sequence diagrams 1 for readability
(diagrams are attached in zip)

UML State Machine Diagram 1

TODO: Add labels to the state machine diagrams & update when we get use cases



UML State Machine Diagram 2



Requirements Traceability matrix

ID	Requirement	Related Use Case	Design Element(s)	Implementation element	Test
1	<paragraph 1> The battery indicator shall: - Display in upper-left corner of home screen - Show percentage (0-100%) numerically - Update every 5 minutes or when charging	UC-02	ScreenHome, InsulinPump, MainWindow	MainWindow.cpp (battery logic), screenhome.cpp (display rendering)	UI Test: Verify low battery behavior

2	<paragraph 1> The insulin fill gauge shall: - Display in upper-right corner - Show remaining insulin as bar and numeric value	UC-02, UC-08	ScreenHome, InsulinPump,	insulinpump.cpp (level monitoring), screenhome.cpp (visual display)	System Test: Verify all warning thresholds
3	<paragraph 1> The IOB indicator shall: - Display active iob - Show numeric value - Include countdown timer (HH:MM format)	UC-02, UC-06	ScreenHome, InsulinPump	insulinpump.cpp (decay algorithm), screenhome.cpp (timer display)	Unit Test: Verify IOB decay calculations
4	<paragraph 1> The bolus button shall: - Be located center on home screen - Display "BOLUS" - Immediately transition to bolus calculator on press	UC-04	ScreenHome, ScreenBolus	mainwindow.cpp (navigation logic), screenhome.cpp (button states)	UI Test: Verify all button states and transitions
5	<paragraph 3> Profile creation shall require: - Basal rate - Carb ratio - Correction - Target BG - Time range	UC-03	ProfileManager, PersonalProfile, ScreenAddProfile	profilemanager.cpp (input validation), personalprofile.h (data storage)	Unit Test: Boundary value analysis for all parameters
6	<paragraph 3> Profile management shall: - Enforce maximum 6 profiles - Require unique names	UC-03	ProfileManager, ScreenProfileSetup	profilemanager.cpp, screenprofilesetup.cpp	Unit Test: Verify profile limit enforcement,

	(case-sensitive) - CRUD capabilities				CRUD
7	<paragraph 4> Bolus calculation shall: - Use bolus calculation formula - Store calculation history	UC-04	ScreenHome, ScreenBolus, InsulinPump, BolusCalculator.	mainwindow.cpp (navigation logic), screenhome.cpp (button states), insulinpump.cpp(data translation), boluscalculator.cpp (bolus calculations)	Unit Test: Verify rounding and capping behavior
8	<paragraph 4> Extended bolus shall: - Supports extended bolus calculations and immediate bolus calculation depending on input - Allow cancellation - Log interrupted deliveries	UC-04	BolusCalculator, MainWindow, ScreenBolus	boluscalculator.cpp (extended delivery logic), mainwindow.cpp, screenbolus.cpp	System Test: Verify all extended delivery scenarios
9	<paragraph 5> Low glucose suspension shall: - Trigger at <3.9 mmol/L - Display "LOW GLUCOSE SUSPEND"	UC-05	InsulinPump, Error, ScreenHome	insulinpump.cpp (safety checks)	Safety Test: Verify all suspension conditions
10	<paragraph 5> High glucose alerts shall: - Trigger at >10 mmol/L - Deliver a fixed insulin delivery	UC-06	InsulinPump, Error	insulinpump.cpp (alert system)	System Test: Verify alert timing and persistence
11	<paragraph 5> Auto-resume functionality shall:	UC-05	InsulinPump, ScreenProfileSetup, MainWindo	insulinpump.cpp (resume logic), screenprofile.cpp, mainwindow.cpp	Test: Verify auto resume function

	-Automatically switch profile if current profile is complete - Allow manual override of delay - Log all resume attempts		w		
12	<paragraph 6> Data visualization shall: - Display glucose trends Via graph - Overlay insulin delivery events - Show high/low glucose zones	UC-06	Chart, InsulinPump, MainWindow	chart.cpp(graph rendering), insulinpump.cpp, mainwindow.cpp	UI Test: Verify data visualised
13	<paragraph6>All the information is displayed in the log console. -Shows recent events. -Helps track glucose patterns. -Provides previous data.	UC-06	MainWindow	mainwindow.cpp	Test : displaying proper outputs
14	<paragraph 7> Low insulin handling shall: - Trigger alert at ≤2 units remaining -when insulin is out don't deliver any bolus or basal	UC-07	InsulinPump, Error	insulinpump.cpp (supply monitoring)	Safety Test: Verify warnings are made and deliveries are halted

Implementation clarifications:

We assume there is only one profile(hence no feature to switch profiles) time periods are different user profiles(morning, afternoon, night), e.g. If you activate an afternoon profile in the morning, it will not activate until afternoon, maybe if you’re about to eat a large meal you activate a certain profile for eating

It can be time based which automatically turns on or based on situations(exercising, sleeping, eating, etc) which requires manual input
Our implementation doesn't have multiple time segments, only one time range per profile(24 hour clock)
We edit the time from main(shown in home page)
function calls across different screens(in the frame) are done through signals
Users can update all aspects of a profile except for its name, time editing needs to be exact, following the format:(XX:XX-XX:XX)
There can only be a maximum of 6 number of profiles in the profile table
Attempting to add a profile with a preexisting name will not register, the contents of the original will remain the same
When entering manual Bolus, a selected CGM is needed as to get the target BG, the manual glucose is optional when viewing calculations(any input that <= 0 will be considered invalid and thus taken from the active profile)
Only one profile/bolus can delivered at a time
A log class was not implemented, instead a log console in the main window shows all the basal, bolus, time, safety events, etc(we did not include correction factor as we thought it would be too text heavy)

Video:

https://drive.google.com/file/d/1NPc98Sf6OeeML360bttzO09-hX6fyo8J/view?usp=sharing