Implementation and Assumption Notes

This project implements a simple prototype device interface using Qt with Python (PySide6) connected to a pseudo sensor that generates simulated humidity and temperature readings. The UI is event-driven: button presses in the interface trigger actions in the main program (such as reading sensor data, performing batch reads, computing statistics, or exiting). Sensor values are displayed in real time through labels and progress bars, with alarms triggered when thresholds are exceeded.

System Implementation

- Language & Framework: Python 3.12 with PySide6 for the graphical interface.
- Pseudo Sensor: A custom `PseudoSensor` class produces humidity values in the range 0–100% and temperature values in the range −20 to 100 °F, with small random variations to simulate sensor noise.
- Database Storage: A lightweight SQLite database (`readings.db`) is used to persist each sensor reading along with a timestamp. The timestamp is stored as a Unix epoch value and displayed in human-readable local time when records are queried.
- UI Features:
    - *Read 1*: Retrieves a single reading, stores it with timestamp, displays it in the UI, and checks alarms.
    - *Read 10*: Collects ten sequential readings at 1-second intervals, each with timestamp storage and live updates.
    - *Statistics*: Computes minimum, maximum, and average for the most recent 10 (or fewer) records.
    - *Show Last 10 Records*: Displays the last 10 stored readings with timestamps in a scrollable dialog.
    - *Alarm System*: User-adjustable thresholds for humidity and temperature; any value exceeding these thresholds triggers a red "ALARM!" indicator.
    - *Exit*: Gracefully closes the database connection and application.
- UI Design: Emphasizes clarity with grouped sections (live readings, alarm thresholds, control buttons) and feedback labels.

Assumptions and Adjustments

- Sensor Range: The pseudo sensor was adapted to output Fahrenheit (−20 to 100 °F) rather than Celsius, as required by the assignment.
- Database Choice: SQLite was chosen for simplicity, portability, and no external dependencies, but the schema is extendable to MariaDB/MySQL if required.
- Alarms: Default thresholds were set at 85% humidity and 90 °F temperature; these values can be adjusted by the user at runtime.
- Batch Timing: Batch reads are handled with a Qt `QTimer` to ensure accurate one-second intervals without blocking the UI.
- Timestamps: Implemented as Unix epoch values stored in the database, and displayed in human-readable format (`YYYY-MM-DD HH:MM:SS`) when records are shown.
- Limitations: Only the most recent 10 readings are displayed or used in statistics, as specified; older records remain in the database but are not visualized.

Issues Encountered

- Windows Virtual Environment: Running and activating Python virtual environments in PowerShell required bypassing execution policy and calling the interpreter directly (e.g., `.venv\Scripts\python.exe main.py`).
- Timestamp Visibility: Although timestamps were stored correctly from the beginning, additional UI elements (a "Last Reading" label and "Show Last 10 Records" viewer) were added to make them explicitly visible to the user.

Code

A simple pseudo-sensor that produces humidity (%) and temperature (°F) values within the assignment's required ranges, with slight random jitter to mimic real sensor noise.

Assignment ranges:
- Humidity: 0–100 %
- Temperature: -20–100 °F

pseudo_sensor.py

```
import random

class PseudoSensor:
   # Discrete "bands" to emulate environmental shifts over time. Each call
   # jittters within the current band, then advances to the next band.
   # Humidity bands (percent)
   h_range = [0, 20, 20, 40, 40, 60, 60, 80, 80, 90, 70, 50, 30, 10, 0]
   # Temperature bands (Fahrenheit)
   t_range = [-20, -10, 0, 10, 30, 50, 70, 85, 95, 100, 90, 70, 50, 30, 10]

  # Position within the bands
   def __init__(self):

      self.h_index = 0
      self.t_index = 0
      # Current values (initialized from first band)
      self.hum_val = self.h_range[self.h_index]
      self.temp_val = self.t_range[self.t_index]

   # Produce one pseudo reading (humidity %, temperature °F).

   def generate_values(self):

      Returns
      (h, t) : tuple(float, float)
         h in [0, 100], t in [-20, 100]

      # Add small random jitter to each band to mimic sensor noise
      self.hum_val = self.h_range[self.h_index] + random.uniform(0, 10)
      self.temp_val = self.t_range[self.t_index] + random.uniform(0, 5)
```

```
        # Advance bands (wrap around at end)
        self.h_index = (self.h_index + 1) % len(self.h_range)
        self.t_index = (self.t_index + 1) % len(self.t_range)

        # Clamp to assignment-specified ranges
        h = max(0.0, min(self.hum_val, 100.0))
        t = max(-20.0, min(self.temp_val, 100.0))
        return h, t
```

Qt UI (PySide6) that:
- Responds to button events (Read 1, Read 10, Show Stats, Show Last 10, Exit)
- Displays humidity (%) and temperature (°F)
- Stores each reading to SQLite with a Unix epoch timestamp
- Computes min/max/avg over the last up to 10 readings
- Supports user-set alarm thresholds for humidity and temperature
- Clearly shows the timestamp (human-readable) of the latest reading

Notes:
- Using SQLite keeps everything self-contained but can be swapped for MariaDB/MySQL.
- We avoid blocking the UI: the 10 reads use a QTimer (1-second interval).

main.py

```
import sys
import time
import sqlite3
from statistics import mean
from typing import List, Tuple

from PySide6.QtCore import Qt, QTimer
from PySide6.QtWidgets import (
    QApplication, QWidget, QLabel, QPushButton, QVBoxLayout, QHBoxLayout,
    QGroupBox, QDoubleSpinBox, QProgressBar, QMessageBox, QDialog, QPlainTextEdit
)

from pseudo_sensor import PseudoSensor

DB_PATH = "readings.db"  # SQLite DB file created in working directory

#Format a Unix epoch (seconds) as local time string.
```

```
def fmt_ts(ts_epoch: float) -> str:
   return time.strftime("%Y-%m-%d %H:%M:%S", time.localtime(ts_epoch))

# Main Qt window: groups live readings, alarm controls, and action buttons.
class SensorApp(QWidget):
    """
   Buttons:
    - Read 1:       Single sample -> store (ts,h,t) -> update UI -> alarm check
    - Read 10:      Ten samples at 1s intervals via QTimer; each stored & shown
    - Show Last 10:  Scrollable view of last up to 10 records with timestamps
    - Show Stats:    Min/Max/Avg over last up to 10 readings
    - Exit:         Graceful shutdown (DB close + window close)
    """

   def __init__(self):
      super().__init__()
      self.setWindowTitle("Humidity/Temperature Monitor – Assignment Build")
      self.resize(720, 500)

      #  sensor & database setup
      self.sensor = PseudoSensor()
      # SQLite is embedded and easy to ship; schema is trivial to port to MariaDB/MySQL.
      self.conn = sqlite3.connect(DB_PATH)
      self._init_db()

      #  live reading labels
      self.h_label = QLabel("Humidity: -- %")
      self.t_label = QLabel("Temperature: -- °F")
      for lbl in (self.h_label, self.t_label):
         lbl.setAlignment(Qt.AlignCenter)
         lbl.setStyleSheet("font-size: 18px; font-weight: 600;")

      # Show the human-readable timestamp of the most recent stored reading
      self.ts_label = QLabel("Last reading at: —")
      self.ts_label.setAlignment(Qt.AlignCenter)
      self.ts_label.setStyleSheet("color:#555;")

      #  visual bars for quick feedback
      self.h_bar = QProgressBar(); self.h_bar.setRange(0, 100)
      self.t_bar = QProgressBar(); self.t_bar.setRange(-20, 100); self.t_bar.setTextVisible(False)

      # alarm thresholds with sensible defaults
      self.h_alarm = QDoubleSpinBox()
      self.h_alarm.setRange(0.0, 100.0); self.h_alarm.setValue(85.0)
```

```
self.h_alarm.setSuffix(" %")

self.t_alarm = QDoubleSpinBox()
self.t_alarm.setRange(-20.0, 100.0); self.t_alarm.setValue(90.0)
self.t_alarm.setSuffix(" °F")

# Alarm indicator label: green when safe, red when an alarm is active
self.alarm_label = QLabel("No alarms")
self._set_alarm_state(False)

# Status line shows progress (e.g., during Read 10)
self.status_label = QLabel("Ready.")
self.status_label.setAlignment(Qt.AlignCenter)

# control buttons
self.btn_read_one = QPushButton("Read 1")
self.btn_read_ten = QPushButton("Read 10 (1s apart)")
self.btn_stats = QPushButton("Show Last 10 Stats")
self.btn_show_last10 = QPushButton("Show Last 10 Records")
self.btn_exit = QPushButton("Exit")

# Connect buttons to handlers
self.btn_read_one.clicked.connect(self.read_one)
self.btn_read_ten.clicked.connect(self.read_ten)
self.btn_stats.clicked.connect(self.show_stats)
self.btn_show_last10.clicked.connect(self.show_last10_records)
self.btn_exit.clicked.connect(self.exit_app)

# Timer used for the 10-sample batch at 1-second intervals (non-blocking)
self.batch_timer = QTimer(self)
self.batch_timer.setInterval(1000)  # 1 second
self.batch_timer.timeout.connect(self._batch_read_tick)
self.batch_remaining = 0  # counts down from 10 during batch

# layout: group boxes + rows keep the UI clear
top = QGroupBox("Live Readings")
top_layout = QVBoxLayout()
top_layout.addWidget(self.h_label)
top_layout.addWidget(self.h_bar)
top_layout.addWidget(self.t_label)
top_layout.addWidget(self.t_bar)
top_layout.addWidget(self.ts_label)  # show latest timestamp
top.setLayout(top_layout)
```

```
    alarms = QGroupBox("Alarm Thresholds")
    alarms_layout = QHBoxLayout()
    alarms_layout.addWidget(QLabel("Humidity alarm:"))
    alarms_layout.addWidget(self.h_alarm)
    alarms_layout.addSpacing(20)
    alarms_layout.addWidget(QLabel("Temperature alarm:"))
    alarms_layout.addWidget(self.t_alarm)
    alarms_layout.addStretch()
    alarms_layout.addWidget(self.alarm_label)
    alarms.setLayout(alarms_layout)

    buttons_layout = QHBoxLayout()
    buttons_layout.addWidget(self.btn_read_one)
    buttons_layout.addWidget(self.btn_read_ten)
    buttons_layout.addWidget(self.btn_stats)
    buttons_layout.addWidget(self.btn_show_last10)
    buttons_layout.addStretch()
    buttons_layout.addWidget(self.btn_exit)

    root = QVBoxLayout()
    root.addWidget(top)
    root.addWidget(alarms)
    root.addLayout(buttons_layout)
    root.addWidget(self.status_label)
    self.setLayout(root)

# Database
def _init_db(self):
    cur = self.conn.cursor()
    cur.execute
    self.conn.commit()

def _insert_reading(self, h: float, t: float, ts: float | None = None) -> float:
    if ts is None:
        ts = time.time()
    cur = self.conn.cursor()
    cur.execute(
        "INSERT INTO readings (ts, humidity, temperature) VALUES (?, ?, ?)",
        (ts, float(h), float(t))
    )
    self.conn.commit()
    return ts

def _fetch_last_n(self, n: int) -> List[Tuple[float, float, float]]:
```

```
    cur = self.conn.cursor()
    cur.execute(
        "SELECT ts, humidity, temperature FROM readings ORDER BY ts DESC LIMIT ?",
        (n,)
    )
    return cur.fetchall()

#  UI update + alarm logic

def _update_latest_display(self, h: float, t: float, ts: float | None = None):
    h_clamped = max(0, min(int(h), 100))
    t_clamped = max(-20, min(int(t), 100))

    self.h_label.setText(f"Humidity: {h:.1f} %")
    self.t_label.setText(f"Temperature: {t:.1f} °F")
    self.h_bar.setValue(h_clamped)
    self.t_bar.setValue(t_clamped)

    if ts is not None:
        self.ts_label.setText(f"Last reading at: {fmt_ts(ts)}")

    self._check_alarms(h, t)

def _check_alarms(self, h: float, t: float):
    alarm = (h > self.h_alarm.value()) or (t > self.t_alarm.value())
    self._set_alarm_state(alarm)

def _set_alarm_state(self, alarm_on: bool):        if alarm_on:
        self.alarm_label.setText("ALARM!")
        self.alarm_label.setStyleSheet(
            "color: white; background:#c62828; padding:4px 8px; font-weight:700;"
        )
    else:
        self.alarm_label.setText("No alarms")
        self.alarm_label.setStyleSheet(
            "color: #2e7d32; background:#e8f5e9; padding:4px 8px; font-weight:600;"
        )

#  Button handlers
def read_one(self):
    h, t = self.sensor.generate_values()
    ts = self._insert_reading(h, t)  # DB timestamp
    self._update_latest_display(h, t, ts=ts)
    self.status_label.setText("Read 1: done.")
```

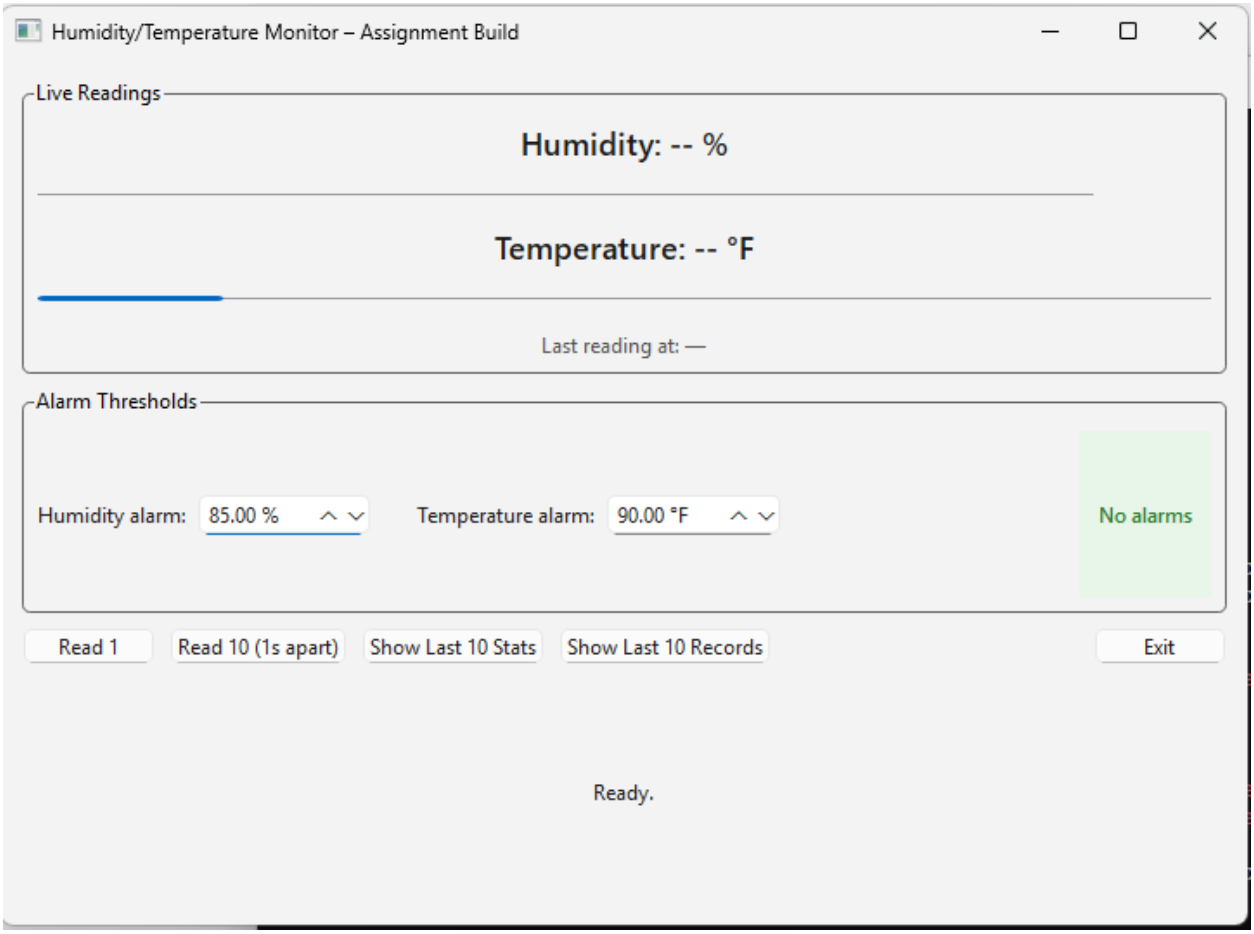
```
def read_ten(self):
    if self.batch_timer.isActive():
        return  # Ignore if already running
    self.batch_remaining = 10
    self.status_label.setText("Starting 10 reads...")
    self.btn_read_one.setEnabled(False)
    self.btn_read_ten.setEnabled(False)
    self.batch_timer.start()

def _batch_read_tick(self):
    h, t = self.sensor.generate_values()
    ts = self._insert_reading(h, t)
    self._update_latest_display(h, t, ts=ts)

    self.batch_remaining -= 1
    self.status_label.setText(f"Reading batch... remaining: {self.batch_remaining}")

    if self.batch_remaining <= 0:
        self.batch_timer.stop()
        self.btn_read_one.setEnabled(True)
        self.btn_read_ten.setEnabled(True)
        self.status_label.setText("Read 10: complete.")

def show_stats(self):
    rows = self._fetch_last_n(10)
    if not rows:
        QMessageBox.information(self, "Stats", "No readings available yet.")
        return

    # rows are newest-first: (ts, humidity, temperature)
    humidities = [r[1] for r in rows]
    temperatures = [r[2] for r in rows]
    newest_ts = rows[0][0]

    msg = (
        f"Stats over last {len(rows)} reading(s) (latest: {fmt_ts(newest_ts)}):\n\n"
        f"Humidity (%)\n"
        f"  Min:  {min(humidities):.1f}\n"
        f"  Max:  {max(humidities):.1f}\n"
        f"  Avg:  {mean(humidities):.1f}\n\n"
        f"Temperature (°F)\n"
        f"  Min:  {min(temperatures):.1f}\n"
        f"  Max:  {max(temperatures):.1f}\n"
```

```
            f"  Avg:  {mean(temperatures):.1f}\n"
        )
        QMessageBox.information(self, "Last 10 Stats", msg)

    def show_last10_records(self):
        rows = self._fetch_last_n(10)
        if not rows:
            QMessageBox.information(self, "Last 10 Records", "No readings available yet.")
            return

        # For readability, display oldest -> newest
        lines = []
        for ts_epoch, h, t in rows[::-1]:
            lines.append(f"{fmt_ts(ts_epoch)}  |  Hum: {h:.1f} %  |  Temp: {t:.1f} °F")

        dlg = QDialog(self)
        dlg.setWindowTitle("Last 10 Records")
        dlg.resize(520, 320)

        layout = QVBoxLayout(dlg)
        view = QPlainTextEdit()
        view.setReadOnly(True)
        view.setPlainText("\n".join(lines))
        layout.addWidget(view)

        ok = QPushButton("Close")
        ok.clicked.connect(dlg.accept)
        layout.addWidget(ok, alignment=Qt.AlignmentFlag.AlignRight)

        dlg.exec()

    def exit_app(self):
        """Gracefully close the DB and exit the app."""
        try:
            self.conn.close()
        except Exception:
            pass
        self.close()

def main():
    app = QApplication.instance() or QApplication(sys.argv)
    win = SensorApp()
    win.show()
    sys.exit(app.exec())
```

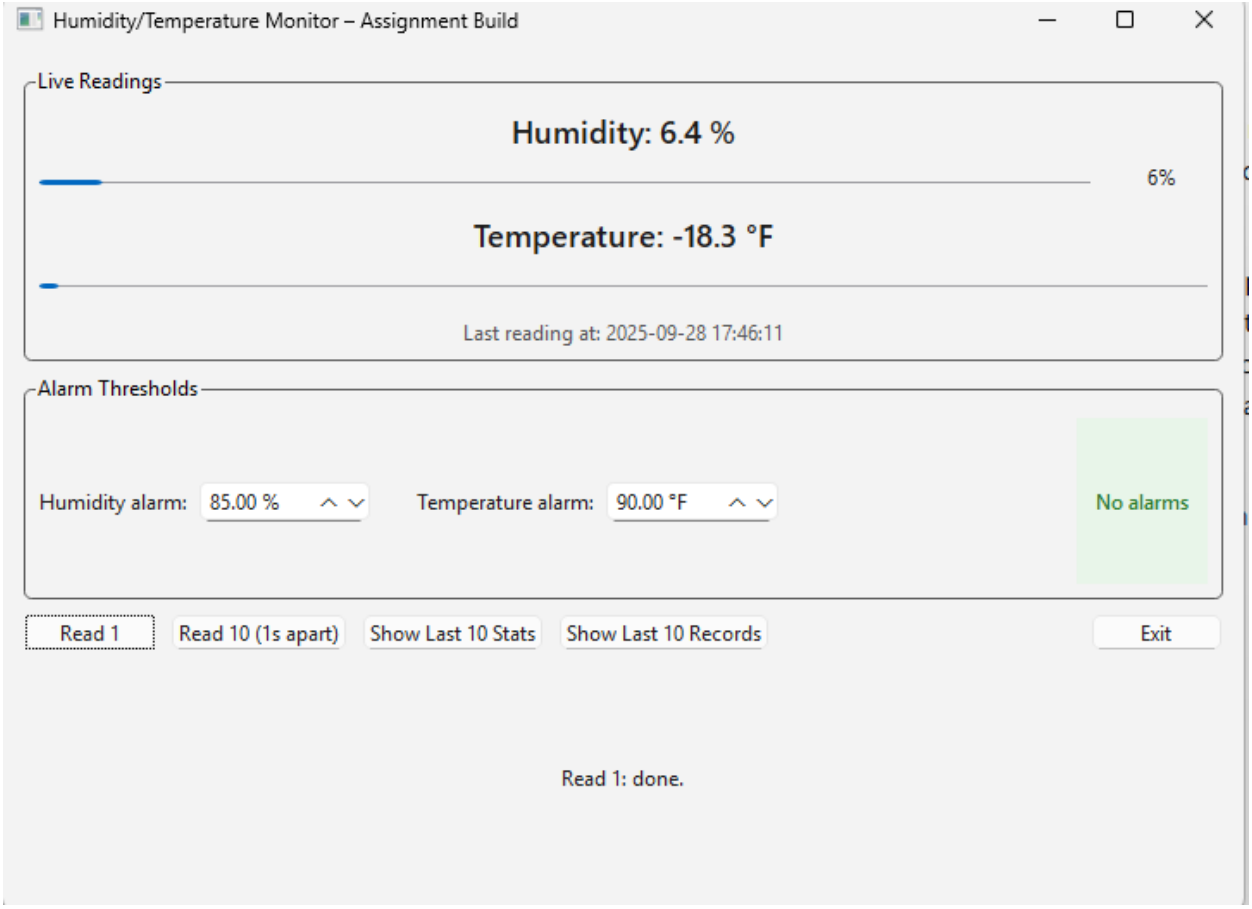
```
if __name__ == "__main__":
    main()
```

Qt UI in action.

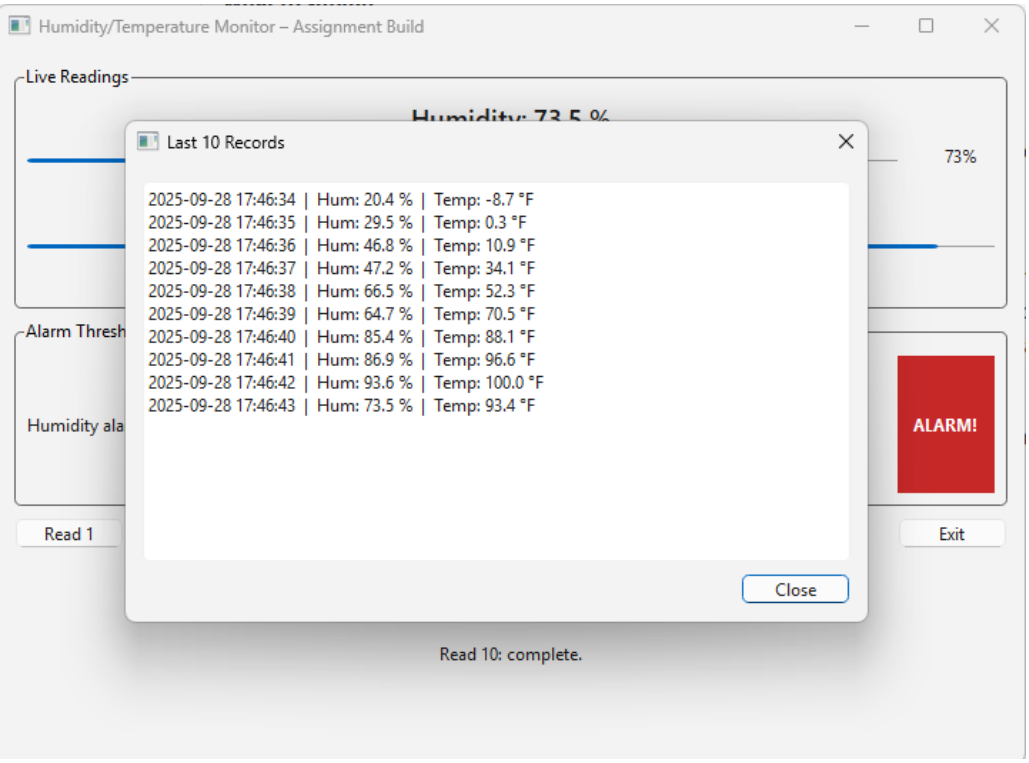
Start Up

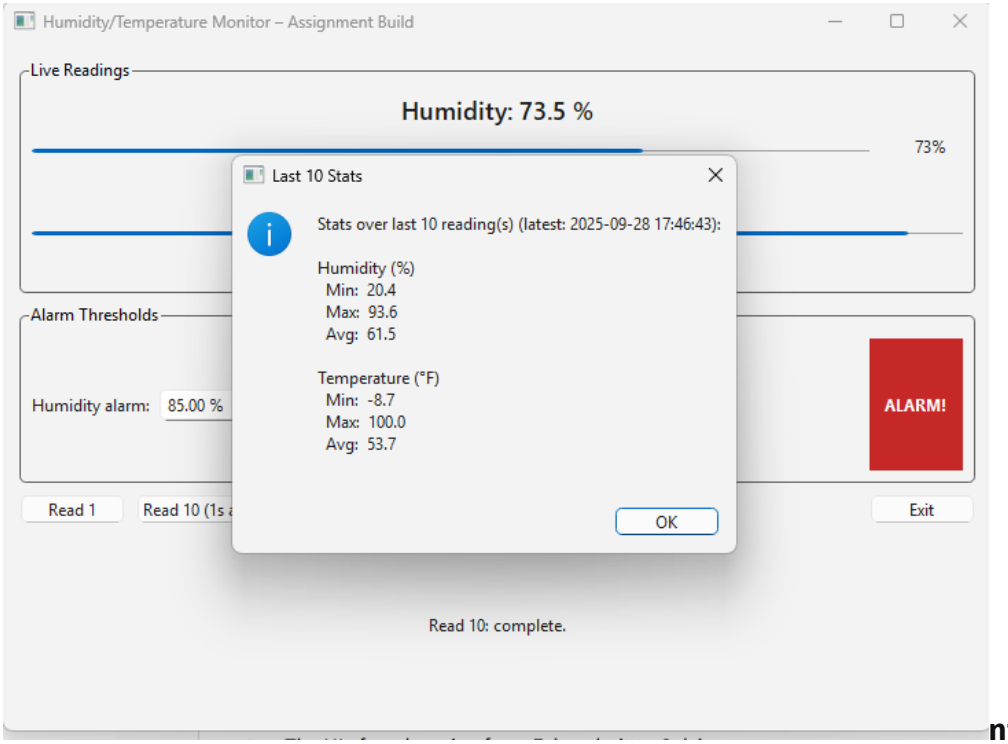


The UI after its first single data point reading



The UI after it has calculated a 10 point Average





nt average

The UI after it has seen either a temperature or humidity alarm (or both)