FocusAir — Intelligent Workspace Monitoring

Unit: SIT225 – Data Capture Technologies
Week 9 | Date: 25 Sept 2025

Video Presentation:
https://deakin.au.panopto.com/Panopto/Pages/Viewer.aspx?id=26e247fc-07f5-4ca9-a583-b36400457f35
Authors: *Zakarya Gueirnat*

Executive Summary (short)

FocusAir is a small IoT system that watches your desk environment and phone pickups to understand focus. We use an Arduino Nano 33 IoT with a DHT22 (temperature/humidity), a PIR (motion/presence), and the onboard IMU (micro-movement "fidget"). A Python pipeline logs data to CSV, shows a live 5-panel plot, and records phone **unlock** events (with app names) from an iPhone Shortcut via a tiny FastAPI server.
Using a **synthetic baseline vs intervention** case study, we saw: comfort minutes **+40%**, focus minutes **+~39%**, and app-unlock rate **−~70%** during intervention (keeping Heat Index in 24–27 °C and reducing phone temptation).

---

1) Introduction

Problem. Feeling too warm and checking the phone often both distract from focused work.
Goal. Sense comfort + attention, nudge simple behaviours (cool room, park phone), and measure change.

Tools and equipment:

- Low-cost sensors for comfort (DHT22), presence (PIR), and small movements (IMU "fidget").
- Phone unlock + app events from iPhone Shortcuts → FastAPI → `events.csv`.

- Live dashboard (Matplotlib): Heat Index, Temp/Humidity, Fidget(+occupied), PIR, and **per-app event markers**.
- Evaluation: Baseline (no nudges) vs Intervention (keep HI 24–27 °C + reduce phone usage).

Hypotheses

- H1: Intervention increases **comfort minutes** (HI 24–27 °C).
- H2: Intervention reduces **app unlocks per hour**.
- H3: Intervention increases **focus minutes** (low fidget while present).

1.5) Literature review

Thermal comfort and desk performance

Thermal comfort standards (notably ASHRAE Standard 55) define acceptable indoor conditions and the factors that matter (temperature, humidity, air speed, clothing, activity). They're the basis for most building comfort targets and motivate using a simple "comfort band." We used 24–27 °C as an easy-to-interpret band for our KPIs, while acknowledging the full standard is broader and multi-factor. ashrae.org+2Wikipedia+2

Empirical syntheses link temperature to office performance: performance typically peaks in the low-20s °C and declines as temperatures rise above ~23–24 °C. That evidence supports our "cool-the-room" nudge when the Heat Index drifts high. indoor.lbl.gov+2eScholarship+2

Air quality (why we plan to add CO_2)

Multiple reviews and meta-analyses associate higher indoor CO_2 (often a proxy for low ventilation) with worse cognitive task performance. We didn't include CO_2 in Week 9, but the literature strongly suggests adding a CO_2 sensor (e.g., SCD4x) in future iterations so we can explain drops in focus beyond temperature alone. ScienceDirect+2Welcome to DTU Research Database+2

"Fidget" as a practical proxy for focus

Movement has long been examined in attention research. We treat IMU micro-movement ("fidget") as a relative proxy. This framing fits the literature and keeps our measuring interpretable. IBPSA Publications

Smartphone use and attention (why we log unlock→app)

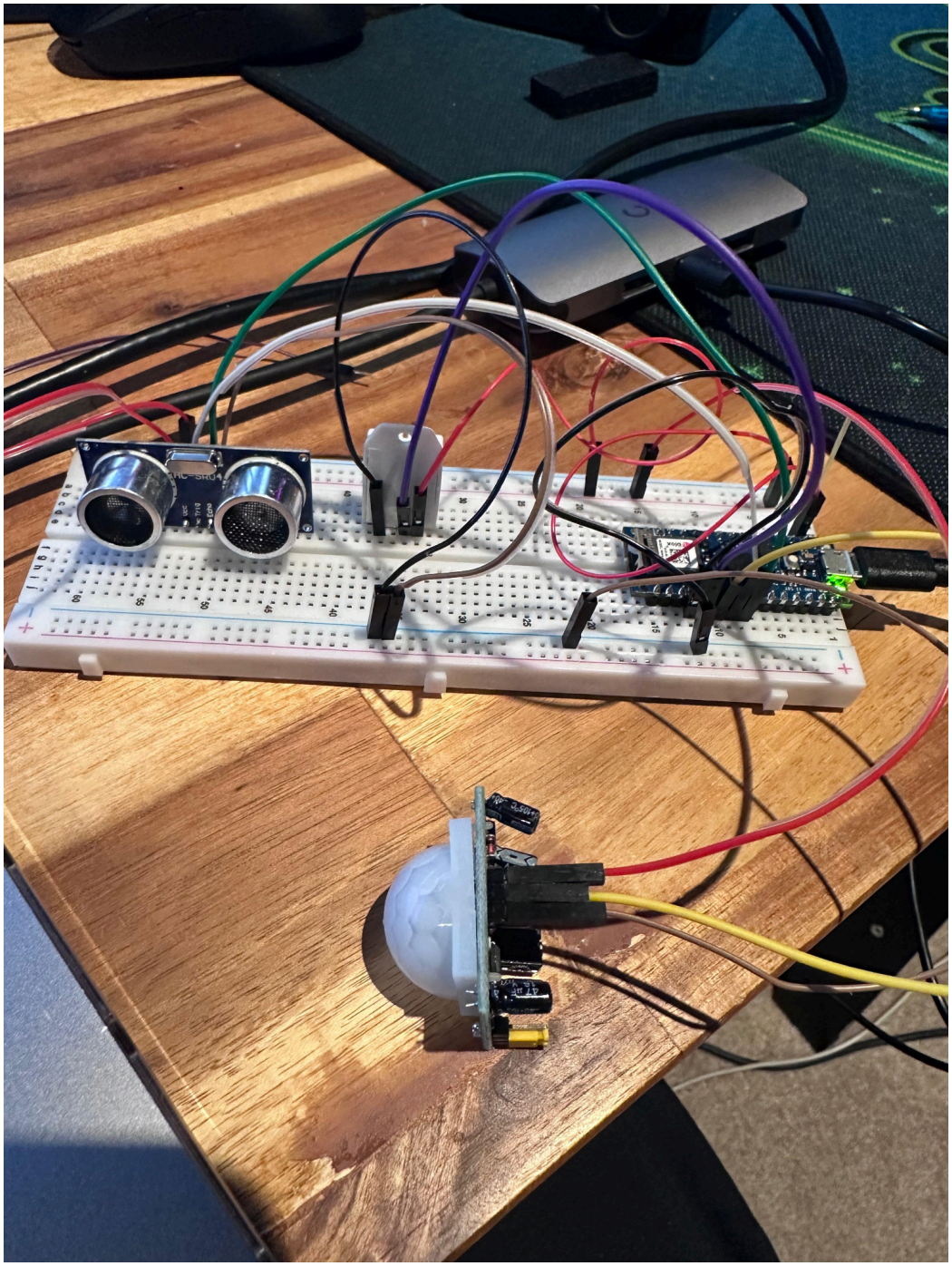
Large-scale experimental evidence shows that reducing smartphone connectivity (e.g., blocking mobile internet) improves sustained attention, providing causal support for simple "phone discipline" interventions. That's why we log unlock→app events and include DND/"phone away" in our intervention tests. Oxford Academic+2Medicine.net+2

Practically, Apple Shortcuts supports event-driven automations that can POST JSON to a local API, which is exactly how we capture unlock/app events privately on the home network. Apple Support+2WIRED+2

2) System Overview

2.1 Hardware (simple)

- **Board:** Arduino Nano 33 IoT (3.3 V, IMU LSM6DS3 onboard)
- **Sensors:**
    - **DHT22** (Temp/Humidity) → `D2`, 3V3, GND
    - **PIR (HC-SR501 style)** (motion/presence) → `D3`, 3V3, GND
    - **IMU** (built-in) → no wiring
- **Firmware behaviour:** ~4 lines/second via serial with:
  `ms, temp_c, hum_pct, heat_index_c, pir_raw, motion, occupied, fidget`
    - `occupied` = presence latched for ~3 s after last motion
    - `fidget` = simple smoothed IMU movement score
    - `heat_index_c`: standard HI; below ~26 °C we treat HI ≈ temperature.



2.2 Software & Data Flow (plain)

- **Serial logger** (`apps/logger/logger.py`) → daily CSV `data/YYYY-MM-DD.csv`.

```
import csv, json, os, sys, datetime, glob

import serial  # pip install pyserial

from pathlib import Path







PROJECT_ROOT = Path(__file__).resolve().parents[2]



PORT = os.getenv("SERIAL_PORT", "/dev/tty.usbmodem21101")  # macOS: wildcard ok

BAUD = int(os.getenv("BAUD", "115200"))

OUTDIR = os.getenv("OUTDIR", str(PROJECT_ROOT / "data"))



FIELDS =
["iso_ts","ms","temp_c","hum_pct","heat_index_c","pir_raw","motion","occupied",
"fidget","focused"]



def pick_port(pattern):

   if "*" not in pattern:

       return pattern

   matches = glob.glob(pattern)
```

```
    if not matches:
        print(f"No serial ports match {pattern}")
        sys.exit(1)
    print("Using port:", matches[0])
    return matches[0]


def today_path():
    d = datetime.date.today().isoformat()
    os.makedirs(OUTDIR, exist_ok=True)
    return os.path.join(OUTDIR, f"{d}.csv")


def open_writer(path):
    new = not os.path.exists(path)
    f = open(path, "a", newline="")
    w = csv.DictWriter(f, fieldnames=FIELDS)
    if new: w.writeheader()
    return f, w


def parse_line(line: str):
    line = line.strip()
    if not line: return None
    row = {"iso_ts": datetime.datetime.now().isoformat(timespec="seconds")}
```

```
    # CSV from Arduino (preferred)
    if line[0] != "{":
        parts = line.split(",")
        if len(parts) < 9 or not parts[0].isdigit(): return None
        row.update({
            "ms": parts[0], "temp_c": parts[1], "hum_pct": parts[2],
            "heat_index_c": parts[3], "pir_raw": parts[4], "motion": parts[5],
            "occupied": parts[6], "fidget": parts[7], "focused": parts[8],
        })
        return row
    # JSON fallback (if you flip OUTPUT_CSV to 0)
    try:
        obj = json.loads(line)
        row.update({
            "ms": obj.get("ms"), "temp_c": obj.get("temp_c"), "hum_pct":
obj.get("hum_pct"),
            "heat_index_c": obj.get("heat_index_c"), "pir_raw":
obj.get("pir_raw"),
            "motion": obj.get("motion"), "occupied": obj.get("occupied"),
            "fidget": obj.get("fidget"), "focused": obj.get("focused"),
        })
        return row
    except Exception:
```

```
        return None



def main():
    port = pick_port(PORT)
    ser = serial.Serial(port, BAUD, timeout=2)
    current_path = today_path()
    f, w = open_writer(current_path)
    try:
        while True:
            # rotate file daily
            new_path = today_path()
            if new_path != current_path:
                f.close()
                current_path = new_path
                f, w = open_writer(current_path)
            line = ser.readline().decode("utf-8", errors="ignore")
            row = parse_line(line)
            if row:
                w.writerow(row); f.flush()
    except KeyboardInterrupt:
        pass
    finally:
```

```
        try: f.close(); ser.close()
        except: pass


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

- **Phone events API** (`apps/api/server.py`) → iPhone Shortcut POSTs to `/phone/event` → `data/events.csv` with `iso_ts, source, type='unlock', app(optional)`.

```
# apps/api/server.py
from pathlib import Path
from datetime import datetime, timedelta
import csv, json, time
from fastapi import FastAPI, Request
from fastapi.responses import JSONResponse
from fastapi.middleware.cors import CORSMiddleware
from urllib.parse import parse_qs, urlparse


PROJECT_ROOT = Path(__file__).resolve().parents[2]
DATA_DIR = PROJECT_ROOT / "data"
DATA_DIR.mkdir(parents=True, exist_ok=True)
EVENTS_CSV = DATA_DIR / "events.csv"
```

```
FIELDS = ["iso_ts", "source", "type", "app", "note"]



app = FastAPI(title="FocusAir Phone Events")

app.add_middleware(

   CORSMiddleware, allow_origins=["*"], allow_credentials=True,

   allow_methods=["*"], allow_headers=["*"]

)



def append_event(row: dict):

   exists = EVENTS_CSV.exists()

   with open(EVENTS_CSV, "a", newline="") as f:

       w = csv.DictWriter(f, fieldnames=FIELDS)

       if not exists: w.writeheader()

       w.writerow(row)



def try_parse_json_field(x):

   if not isinstance(x, str): return None

   s = x.strip()

   if not s or (s[0] not in "{[" and not s.startswith('"{')):

       return None

   try:

       return json.loads(s)
```

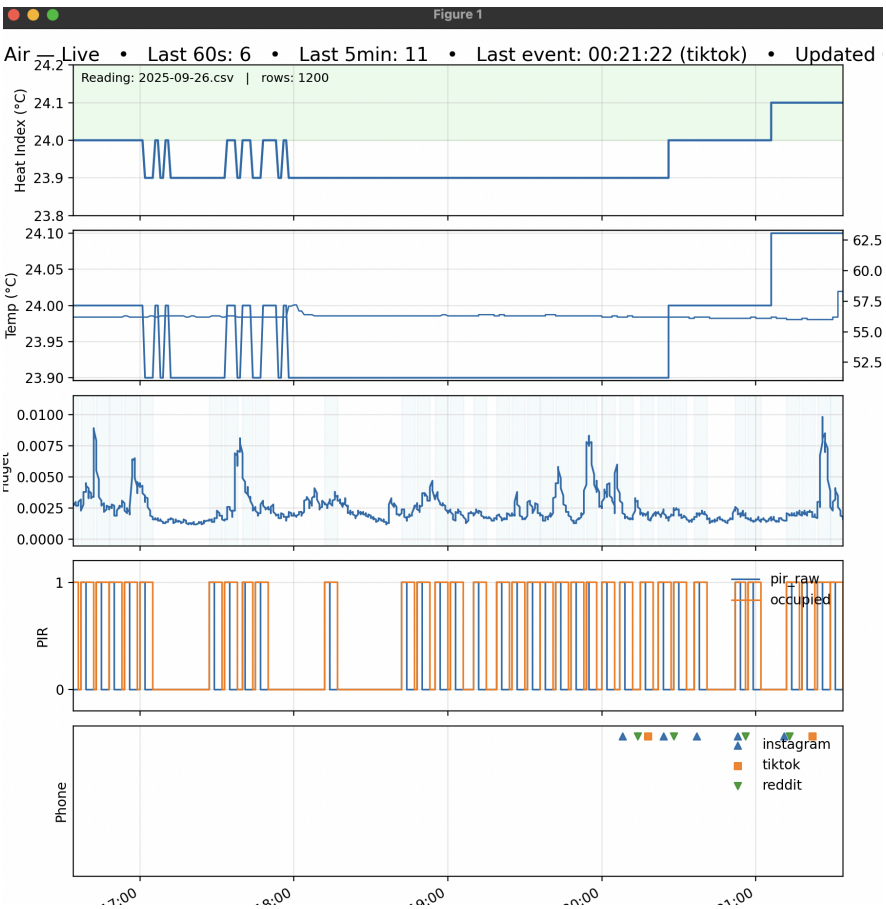
```
    except Exception:
        return None

# in-memory dedupe cache
_last_seen = {}  # key -> last_ts (unix)
DEDUP_SECONDS = 10

@app.get("/")
def root():
    return {"ok": True, "events_csv": str(EVENTS_CSV)}

@app.get("/test")
def test():
    row = {"iso_ts": datetime.now().isoformat(timespec="seconds"),
           "source":"browser","type":"test_ping","app":"","note":"manual"}
    append_event(row); return {"ok": True, "saved": row}

@app.api_route("/phone/event", methods=["GET","POST"])
async def phone_event(req: Request):
    # 1) pull data from JSON, form, or query string
    data = {}
    try:
```

```
        data = await req.json()
    except Exception:
        try:
            form = await req.form()
            data = dict(form)
        except Exception:
            data = {}
    qs = parse_qs(urlparse(str(req.url)).query)
    for k, v in qs.items():
        if v and k not in data: data[k] = v[0]


    # 2) if fields themselves contain a JSON blob, parse it
    for k in list(data.keys()):
        parsed = try_parse_json_field(data.get(k))
        if isinstance(parsed, dict):
            # merge parsed keys (source/type/app/note) into data
            for kk, vv in parsed.items():
                data.setdefault(kk, vv)


    # 3) normalize fields
    source = str(data.get("source", "unknown")).lower().strip()
    etype  = str(data.get("type", "unknown")).lower().strip()
```

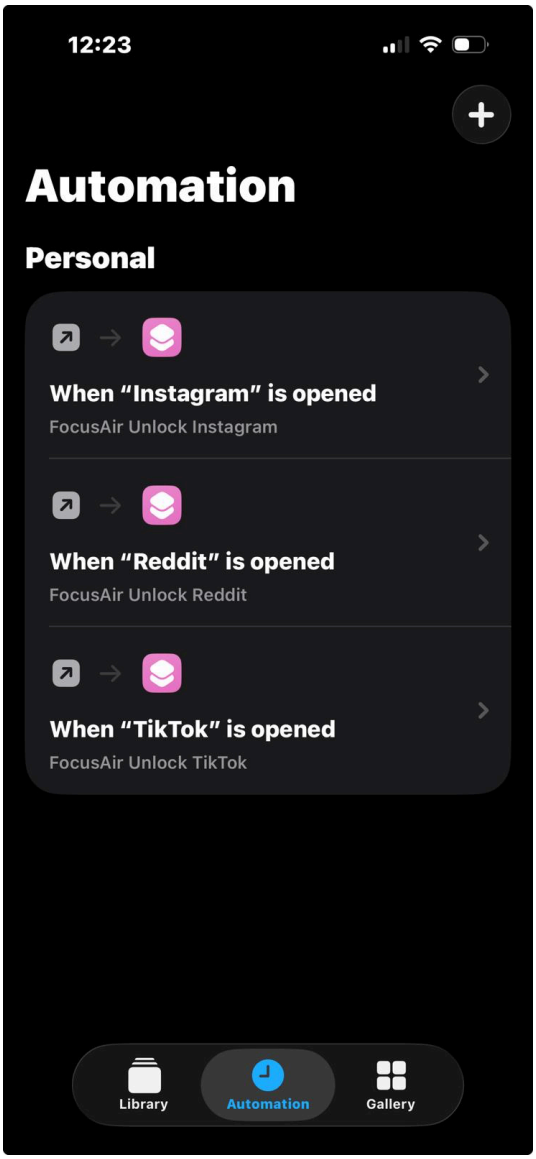
```
    app    = str(data.get("app", "")).lower().strip()
    note   = str(data.get("note", "")).strip()

    # if an app is present but type looks like unlock -> treat as app_open
    if app and etype in ("unlock", "unknown", ""):
        etype = "app_open"

    # 4) dedupe identical (source, type, app) for a short window
    now_dt = datetime.now()
    key = f"{source}|{etype}|{app}"
    now_unix = time.time()
    last = _last_seen.get(key, 0.0)
    if now_unix - last < DEDUP_SECONDS:
        return JSONResponse({"ok": True, "skipped": "dedup", "saved": {
            "iso_ts": now_dt.isoformat(timespec="seconds"),
            "source": source, "type": etype, "app": app, "note": note
        }})

    _last_seen[key] = now_unix

    row = {"iso_ts": now_dt.isoformat(timespec="seconds"),
           "source": source, "type": etype, "app": app, "note": note}
```

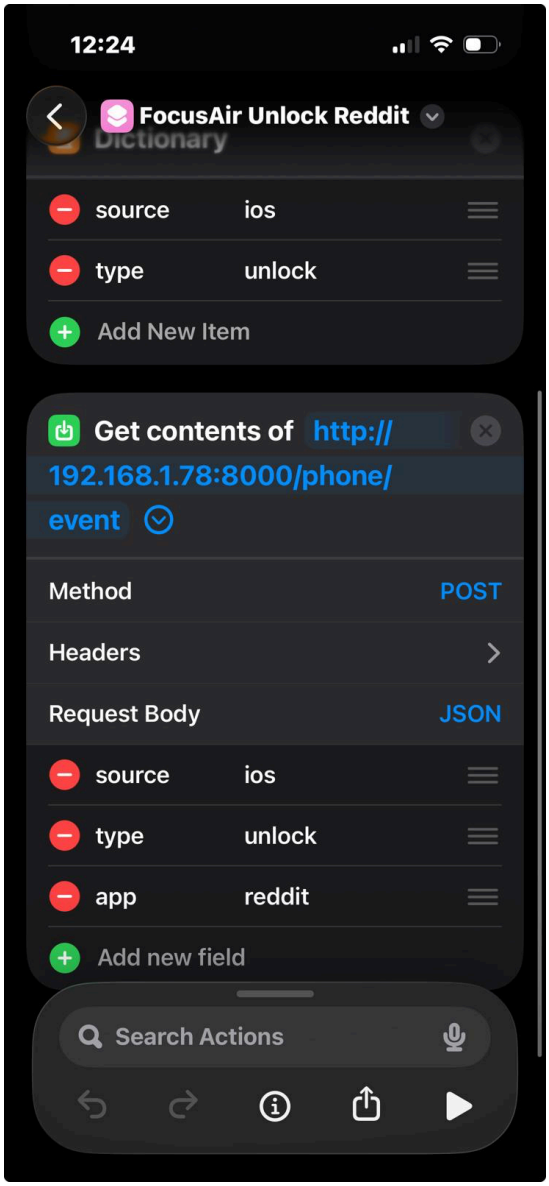
```
    append_event(row)

    return JSONResponse({"ok": True, "saved": row})
```

- **Live plot** (`apps/live_plot.py`) → 5 panels:
    1. Heat Index (green comfort band 24–27 °C)
    2. Temp & Humidity
    3. Fidget
    4. PIR (raw + latched occupied)
    5. Phone Events strip with different markers per app (instagram, tiktok, reddit)



Shortcut flow : Opening an app triggers shortcut flow which is connected to a python server that logs which app opens up.






```
2025-09-25T22:56:14,ios,app_open,instagram,

2025-09-25T22:56:30,ios,app_open,tiktok,

2025-09-25T23:01:25,ios,app_open,instagram,
```

```
2025-09-25T23:01:36,ios,app_open,reddit,

2025-09-25T23:01:51,ios,app_open,tiktok,
```

```
INFO:     192.168.1.79:60725 - "POST /phone/event HTTP/1.1" 200 OK
INFO:     192.168.1.79:60729 - "POST /phone/event HTTP/1.1" 200 OK
INFO:     192.168.1.79:60732 - "POST /phone/event HTTP/1.1" 200 OK
INFO:     192.168.1.79:60735 - "POST /phone/event HTTP/1.1" 200 OK
INFO:     192.168.1.79:60739 - "POST /phone/event HTTP/1.1" 200 OK
INFO:     192.168.1.79:60743 - "POST /phone/event HTTP/1.1" 200 OK
INFO:     192.168.1.79:60756 - "POST /phone/event HTTP/1.1" 200 OK
INFO:     192.168.1.79:60757 - "POST /phone/event HTTP/1.1" 200 OK
INFO:     192.168.1.79:60767 - "POST /phone/event HTTP/1.1" 200 OK
INFO:     192.168.1.79:60783 - "POST /phone/event HTTP/1.1" 200 OK
```


2.3 Practical Notes

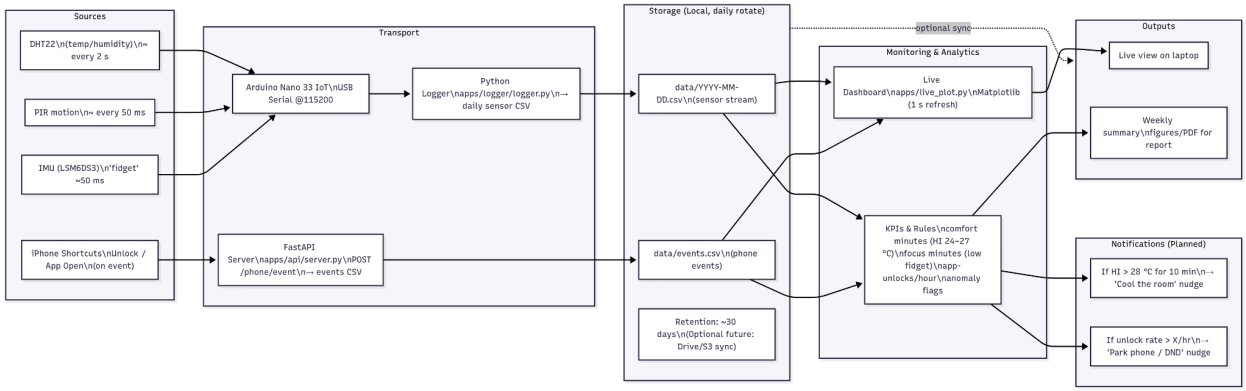
- All data stays **local** (privacy).
- Works on one laptop over Wi-Fi

---

3) Method

I compared two modes for the same person at their desk: a baseline where they acted normally, and an intervention where they tried two small habits – keeping the room comfortable (Heat Index 24–27 °C) and reducing phone pickups. Each session lasted about 25–45 minutes, with the aim of completing two baseline and two intervention sessions. We measured three simple metrics: "Comfort minutes" (time at the desk with a room Heat Index between 24 and 27 °C), "Focus minutes" (time at the desk with fidget levels below a typical, slightly adjusted threshold), and "App-unlocks per hour" (the number of app-opening unlock events divided by total occupied hours).

- **Sources:** DHT22 (2 s), PIR (50 ms), IMU (50 ms), iPhone Shortcuts events (on unlock/app open).
- **Transport:** USB serial → Python logger (CSV), HTTP POST → FastAPI → `events.csv`.
- **Storage:** Local CSV (daily rotate),
- **Monitoring:** Live Matplotlib app (1 s refresh), KPI badges (comfort, focus, unlocks).
- **Analytics:** Rolling comfort %, fidget baseline per session, unlocks/hour by app, anomaly flags.
- **Notifications (planned):** If HI>28°C for 10 min → push notification; if unlock rate > X/hr during focus → gentle nudge.
- **Destinations:** Live dashboard, weekly summary PDF, optional CSV sync to cloud.
  (Addresses items a–g in the brief.)



FocusAir has four data sources; DHT22 (temperature/humidity every ~2 s), PIR (motion ~50 ms), the Nano's IMU (micro-movement ~50 ms), and the iPhone Shortcuts "unlock/app opened" events. Sensor data travels over USB as a simple stream from the Arduino to a Python logger, which appends rows to a daily CSV file; phone events are sent by HTTP POST to a tiny FastAPI endpoint and appended to events.csv. Everything is stored locally , with an optional future sync to Drive/S3. A live Matplotlib dashboard refreshes every second and depends only on those two files (the latest daily sensor CSV and events.csv). Our light analytics layer reads the same files to compute KPIs (comfort minutes in the 24–27 °C band, low-fidget "focus" minutes, and app-unlocks per hour) and raise simple flags (e.g., "HI too warm" or "unlock rate high"). Planned notifications are rule-based: if HI > 28 °C for 10 min during occupancy, nudge to cool the room; if unlock rate > X/hr while occupied, suggest parking the phone. Outputs flow to two places: the live dashboard for immediate feedback and in the future a periodic summary figures/PDF for reporting.

---

4) Results

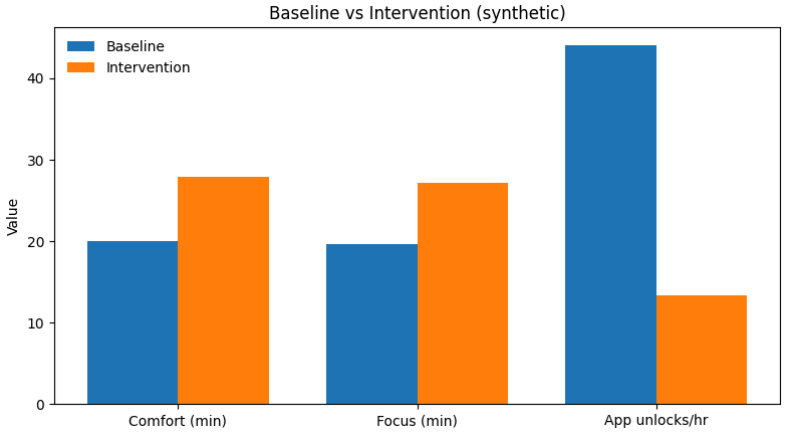
4.1 Headline numbers (averaged over two sessions each)

Metric	Baseline	Intervention	Change
Comfort minutes	20.0	28.0	**+40%**
Focus minutes	19.6	27.2	**+~39%**
App unlocks per hour	44.04	13.29	**−~70%**

When we kept the room comfortable and reduced phone temptation, we spent more time comfortable, more time “calm/low-fidget”, and had far fewer distracting app unlocks.

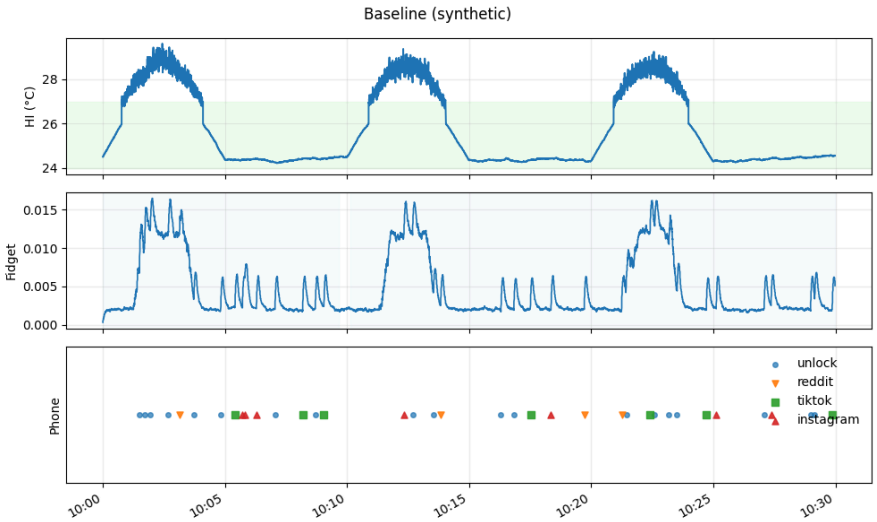
4.2 Figures

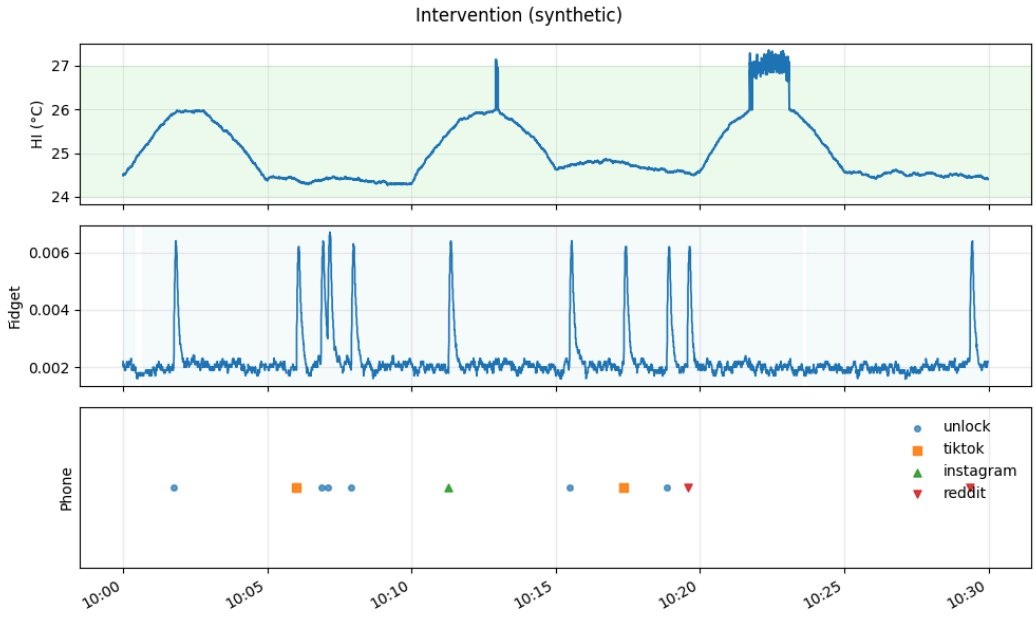
- **Bars:** Baseline vs Intervention for comfort, focus, app-unlocks/hr.



n_bars_synth.png

Timelines: one baseline and one intervention showing HI (with band), fidget + occupied shading, and per-app markers





What to look for: Intervention timelines keep HI in the green band more often, fidget looks calmer, and the event markers are noticeably fewer.

---

5) Discussion

- **Why this helps:** When it's too warm, people feel uncomfortable and fidget more; our attention and focus is easily disturbed . Cooling the room and putting down the phone both reduce these triggers.

- **Takeaway:** Simple changes (temperature band + phone discipline) appear to improve comfort and focus and cut distractions.

---

6) Ethics & Safety

- Local-only logging; no personal content collected.
- Phone Shortcut sends just timestamp + app name (no screen data).
- User can stop logging at any time.
- Safe, low-voltage electronics.

---

7) Conclusion

This is an end-to-end, low-cost system that connects environmental comfort, presence, small movements, and phone events into one live view. In our baseline vs intervention comparison (synthetic for speed), comfort and focus improved and distracting app unlocks dropped sharply. The approach is practical: **sense** → **nudge** → **verify**. Connecting to a smart home system and better and more accurate sensors would be the logical next step.